



DEFENSE TECHNICAL INFORMATION CENTER

Information for the Defense Community

DTIC® has determined on 05/24/2010 that this Technical Document has the Distribution Statement checked below. The current distribution for this document can be found in the DTIC® Technical Report Database.

DISTRIBUTION STATEMENT A. Approved for public release; distribution is unlimited.

© COPYRIGHTED; U.S. Government or Federal Rights License. All other rights and uses except those permitted by copyright law are reserved by the copyright owner.

DISTRIBUTION STATEMENT B. Distribution authorized to U.S. Government agencies only (fill in reason) (date of determination). Other requests for this document shall be referred to (insert controlling DoD office)

DISTRIBUTION STATEMENT C. Distribution authorized to U.S. Government Agencies and their contractors (fill in reason) (date of determination). Other requests for this document shall be referred to (insert controlling DoD office)

DISTRIBUTION STATEMENT D. Distribution authorized to the Department of Defense and U.S. DoD contractors only (fill in reason) (date of determination). Other requests shall be referred to (insert controlling DoD office).

DISTRIBUTION STATEMENT E. Distribution authorized to DoD Components only (fill in reason) (date of determination). Other requests shall be referred to (insert controlling DoD office).

DISTRIBUTION STATEMENT F. Further dissemination only as directed by (inserting controlling DoD office) (date of determination) or higher DoD authority.

Distribution Statement F is also used when a document does not contain a distribution statement and no distribution statement can be determined.

DISTRIBUTION STATEMENT X. Distribution authorized to U.S. Government Agencies and private individuals or enterprises eligible to obtain export-controlled technical data in accordance with DoDD 5230.25; (date of determination). DoD Controlling Office is (insert controlling DoD office).

Proceedings of the Sixth Conference on Computer Generated Forces and Behavioral Representation



July 23-25, 1996
Orlando, Florida
Sponsored by STRICOM•DMSO
Contract - N61339-92-C-0045



DMSO



IST-TR-96-18

Proceedings of the Sixth Conference on Computer Generated Forces and Behavioral Representation

**July 23-25, 1996
Orlando, Florida**

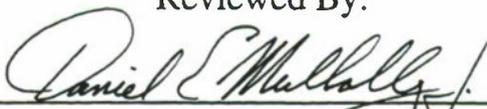
Sponsored by:
Defense Modeling and Simulation Office
U.S. Army Simulation, Training, and Instrumentation Command

Organized by:
Institute for Simulation and Training
3280 Progress Drive
Orlando, Florida 32826

University of Central Florida, Division of Sponsored Research

Contract N61339-92-C-0045 CDRL A00D
IST-TR-96-18

Reviewed By:


Daniel E. Mullally, Jr.



20100311122

Preface

PURPOSE

This report presents the proceedings of the Sixth Conference on Computer Generated Forces (CGF) and Behavioral Representation (BR). The Conference is scheduled from 23 - 25 July in Orlando, Florida and is hosted by the Institute for Simulation and Training (IST). IST is a component of the Division of Sponsored Research at the University of Central Florida.

OBJECTIVES

The objectives of this conference are to:

- Provide a forum for information exchange on CGF and BR modeling research.
- Identify gaps in CGF and BR research.
- Present upcoming research programs and opportunities.
- Present technology demonstrations to the CGF and BR community.

Attendees will have an opportunity to participate in discussions of Service User needs, CGF systems issues, and technical presentations on the components of a CGF.

BACKGROUND

Under the sponsorship of the Defense Modeling and Simulation Office (DMSO, the U.S. Army, Simulation, Training & Instrumentation Command (STRICOM), and the Institute for Simulation and Training of the University of Central Florida is conducting this Sixth Conference on CGF and BR.

UCF/IST has hosted five previous CGF & BR symposia. An indication of the success of these interest group meetings is reflected in the steady attendance, rising from 84 attendees in Oct. 1990 to 128 in May of 1991, to 310 in March of 1993, to 323 in May of 1994 and 281 in May of 1995.

Following the topics outlined in the Second BR symposium, IST is tasked by DMSO and STRICOM to host a continuing series of CGF and BR conferences. These conferences will provide a continuing ability to promote and focus research in this important area. Most attendees at previous conferences expressed an interest in continuing in a dialogue with developers on future requirements in order to justify their own internal research and development participation and commitment to this emerging technology.

Other conference topics which merit consideration for resolution by the community of military, industry, and academic researchers in BR include:

- Interoperability Standards for Behavioral Representation in Defense Simulations;
- Validation, Verification and Accreditation of Behavioral Representation models;
- Functional Specification rationale for Behavioral Representation models in Design, Testing and Training Simulations;
- Interoperability issues for classified modeling in Behavioral Representation;
- Behavioral Representation in Virtual Reality.

GENERAL

This report is presented in one volume. Wherever possible, the papers are arranged in the order of presentation.

A list of attendees will be distributed to all registered attendees at the conclusion of the conference.



Conference Committee

Conference Chair

Daniel E. Mullally, Jr.

Program Committee

Clark R. Karr

Douglas A. Reece

Robert W. Franceschini

James Heusmann

Production Assistance

Doug Barrett

Vicki McGurk

Local Arrangements and Registration

Vince Amico

Karen Gauvin

Linda Hayes

Deodith Mapas

Karen Staaf

Table of Contents

Preface	i
Session 0: Plenary Presentations	
Next Generation Computer Generated Forces.....	3
David R. Pratt <i>Technical Director, Joint Simulation System (JSIMS) Joint Program Office Orlando, FL</i>	
Session 1a: Command Forces Simulation	
Command Forces (CFOR) Status Report.....	11
Susan Hartzog <i>NRAD, NCCOSC RDT&E San Diego, CA</i> Marnie R. Salisbury <i>The Mitre Corporation McLean, VA</i>	
Architecture of a Command Forces Command Entity.....	19
Robert B. Calder, Richard L. Carreiro, James N. Panagos, Rob G. Vrablik, Ben Wise <i>Science Applications International Corp. Burlington, MA</i> Forrest L. Chamberlain, Douglas P. Glasson <i>TASC Reading, MA</i>	
Knowledge Acquisition and Delivery: Constructing Intelligent Software Command Entities.....	31
Seth R. Goldman <i>Hughes Research Labs Malibu, CA</i>	
Task-decomposition Planning for Command Decision Making.....	37
Jonathan Gratch <i>Information Sciences Institute, USC Marina del Rey, CA</i>	

Session 1b: Non-Military Uses of CGF

The CAEN Wargame for OOTW Applications..... 49

Janusz M. Adamson

Defence Evaluation Research Agency

Sevenoaks, Kent, England, UK

MedSAF: Prototyping a Vision for Medical Simulation in DIS..... 57

Anthony J. Courtemanche, Kent Bimson

Science Applications International Corp.

Orlando, FL

A Model of Large-Scale Citizen Evacuation for Emergency Management Simulation..... 67

Ross C. Creech, Mikel D. Petty

Institute for Simulation and Training

Orlando, FL

**Application of Computer Generated Force Technology to Interagency Drug
Interdiction..... 79**

John Miller, Greg Jackson

BMH Associates Inc.

Norfolk, VA

Will Miller

Joint Interagency Task Force East

Key West, FL

Session 2a: Reasoning

Intelligent Agents for Aircraft Combat Simulation..... 93

Silvia Coradeschi, Lars Karlsson, Anders Törne

Linköping University, C&IS

Linköping, Sweden

**The Automated Wingman: An Intelligent Entity for Distributed Virtual
Environments..... 101**

CPT Sheila B. Banks, USAF, Eugene Santos, LTC Martin R. Stytz, USAF

Air Force Institute of Technology

Wright-Patterson, AFB, OH

Moving Intelligent Automated Forces Into Theater-Level Scenarios..... 113

Randolph M. Jones, John E. Laird, Paul E. Nielsen

University of Michigan, AI & Robotics Lab

Ann Arbor, MI

Design of a DIS Agent, the AISim System: A Progress Report..... 119
Sakir Kocabas, Ercan Oztemel, Mahmut Uludag, Nazim Koc
Marmara Research Center
Gebze-Kocaeli, Turkey

Session 2b: Uses of CGF

Computer Generated Forces (CGF) Assessment..... 127
Wilbert J. Brooks, Marguerite M. Dymond
Director, US AMSAA
Aberdeen Proving Grd, MD

Considerations for the Use of Entity-based Simulations for Tactical Decision Making Training..... 131
Jack Berkowitz
The Mitre Corporation
San Diego, CA

Testing Future Weapons Systems Using CGF Systems..... 141
Michael A. Craft, Clark R. Karr
Institute for Simulation and Training
Orlando, FL

Use of ModSAF in Development of an Automated Training Analysis and Feedback System..... 151
Ted Metzler, John Nordyke
LB&M Associates, Inc.
Lawton, OK

Session 3a: Behavior Representation

CCTT SAF and ModSAF Behavior Integration Techniques..... 159
Matthew K. Kraus, Derrick J. Franceschini, Tracy R. Tolley, Lee J. Napravnik,
Daniel E. Mullally, Robert W. Franceschini
Institute for Simulation and Training
Orlando, FL

Semantic Arbitration of Behavior for the Interoperability of SAF Simulations..... 171
Frederic McKenzie, Christopher Dean
Science Applications International Corp.
Orlando, FL
Avelino Gonzalez
Dept. of Elec. and Compt. Engr., UCF
Orlando, FL

Generating Computer Generated Forces.....	181
Robert Balzer <i>Information Sciences Institute, USC</i> <i>Marina del Rey, CA</i>	
A New Mechanism for Cooperative Behavior in ModSAF.....	189
Sumeet Rajput, Clark R. Karr <i>Institute for Simulation and Training</i> <i>Orlando, FL</i>	
Session 3b: Exercise Planning - AAR	
A Briefing-Based Graphical Interface for Exercise Specification.....	203
Karen J. Coulter, John E. Laird <i>University of Michigan, AI & Robotics Lab</i> <i>Ann Arbor, MI</i>	
Scenario and Infrastructure Analysis to Measure Large-Scale CGF Exercise Performance.....	209
Michael Juliano, Robert D’Urso, Ben Wise, Edward Powell <i>Science Applications International Corp.</i> <i>Burlington, MA</i>	
Quickset: A Multimodal Interface for Military Simulation.....	217
James A. Pittman, Ira Smith, Phil Cohen, Sharon Oviatt, Tzu-Chieh Yang <i>Center for Human Computer Communication</i> <i>Oregon Graduate Institute</i> <i>Portland, OR</i>	
Soldier Station: Integrating Constructive and Virtual Models.....	225
David R. Pratt, Shirley Pratt <i>Naval Postgraduate School</i> <i>Monterey, CA</i> David Ohman, John Galloway <i>TRAC</i> <i>White Sands Missile Range, NM</i>	
Session 4a: Learning	
Genetic Algorithms and Force Simulation.....	237
Janusz M. Adamson <i>Defence Evaluation Research Agency</i> <i>Sevenoaks, Kent, England, UK</i> K. G. Joshi <i>EDS Defence Ltd</i> <i>Centrum House, Fleet, Hampshire, England, UK</i>	

Training a ModSAF Command Agent Through Demonstration..... 243

Michael R. Hieb, Gheorghe Tecuci, J. Mark Pullen
George Mason University, CS Dept.
Fairfax, VA

Learning the Selection of Reactive Behaviors..... 255

Sumeet Rajput, Clark R. Karr, Jaime Cisneros
Institute for Simulation and Training
Orlando, FL
Rebecca J. Parsons
University of Central Florida, CS Dept.
Orlando, FL

An Intelligently Interactive Non-Rule-Based Computer Generated Force..... 265

Lawrence J. Fogel, Bill Porto, Mark Owen
Natural Selection, Inc.
La Jolla, CA

Session 4b: Project Status Reports

LeatherNet: A Synthetic Forces Tactical Training System for the USMC

Commander..... 275

Jeff Clarkson
NOSC, NRaD
San Diego, CA
John Yi
KES
San Diego, CA

Computer Generation of Joint Theater Missile Defense (TMD) Assets..... 283

Donald E. Carver, George M. Parsons
US Army Missile Def. PEO: SFAE-MD-TSD-TS
Huntsville, AL
William T. Naff
BDM Federal, Inc.
Huntsville, AL

The JPSD Corps Level Computer Generated Forces (CLCGF) System Project

Update 1996..... 291

Jeffrey C. Peacock, Kevin C. Bombardier, James N. Panagos
Science Applications International Corp.
Burlington, MA
Thomas E. Johnson
Raytheon Company
Sudbury, MA

A Strategic Plan for the Integration of ModSAF and CCTT SAF..... 303
MAJ John D. Norwood
STRICOM, Asst. PM CATT
Orlando, FL

Session 5a: Agent Architecture

Polling vs. Event-driven Computer Generated Forces (CGF) Architectures..... 313
Michael K. Adkins
US Army TRAC
Ft. Leavenworth, KS

Broad Agents for Intelligent Simulation..... 319
Richard T. Hepplewhite, Jeremy W. Baxter
Defence Research Agency, Malvern
Great Malvern, Worcester, UK

Mission Planning and Coordinated Execution for Unmanned Vehicles..... 329
Patrick G. Kenny, Edmund H. Durfee, Karl C. Kluge
University of Michigan, AI & Robotics Lab
Ann Arbor, MI

An Architecture for Computer Generated Individual Combatants..... 337
Douglas A. Reece, Paul Kelly
Institute for Simulation and Training
Orlando, FL

Session 5b: VV&A

ModSAF Credibility..... 347
Ben Paz
STRICOM, AMSTI-EC
Orlando, FL
Irwin L. Hudson
NATIONS, Inc.
Orlando, FL

SAF and Manned Simulators Correlation Issues in CCTT..... 355

Henry Marshall

STRICOM, AMSTI-EE

Orlando, FL

Edward V. Chandler

Science Applications International Corp.

Orlando, FL

Brian R. McEnany

Science Applications International Corp.

McLean, VA

John G. Thomas, Jr.

Director, US AMSAA

Aberdeen Proving Grd, MD

Validation of Individual Combatant Simulation Using a Model-Test-Model

Approach..... 367

George R. Mastroianni

U.S. Army Natick RD&E Center

Natick, MA

Victor E. Middleton

Simulation Tech., Inc.

Dayton, OH

Using the Combat Instruction Set for Verification and Validation of Semi-Automated Force Behaviors: High and Low Intensity Case Studies..... 373

Damon D. Baker, Charles (Chad) W. Mullis

U.S. Army TRAC

White Sands Missile Range, NM

Session 6a: Physical Modeling

Acoustics in Computer Generated Forces..... 381

Robert L. Albright

US Army TRAC

Ft. Leavenworth, KS

Creating a Synthetic Environment for Naval Applications..... 389

Peter B. Howells, G. Giguere

CAE Electronics Ltd.

St. Laurent, Quebec, Canada

Phenomenology Behaviors in ModSAF..... 397

Se-Hung Kwak
Lockheed Martin, ADS
Cambridge, MA
MAJ Reba Lyons
PM DIS - STRICOM
Orlando, FL

Detection Models for Computer Generated Individual Combatants..... 409

Douglas A. Reece, Ralph Wirthlin
Institute for Simulation and Training
Orlando, FL

Session 6b: Systems Architecture

Indirect Fire Support on the ModSAF Virtual Battlefield..... 419

Martin D. Howard
Univ. of Texas at Austin, Applied Research Laboratories
Austin, TX

An Architecture for Linking Aggregate and Virtual Simulations..... 427

Stephen A. Schricker, Robert W. Franceschini, David R. Stober, Jonathan C. Nida
Institute for Simulation and Training
Orlando, FL

Using an Ordnance Server to Provide Validated Weapon Models to ModSAF..... 435

Lawrence Ullom
NAWC-AD, Code 5161, MS-3
Patuxent River NAS, MD
Pete Fischer
J. F. Taylor, Inc.
Lexington Park, MD

Interfacing External Decision Processes to DIS Applications..... 441

Elizabeth L. White, Ken Frosch, Vincent P. Laviano, Michael R. Hieb, J. Mark Pullen
George Mason University, CS Dept.
Fairfax, VA

Session 7a: Individual Combatant Behavior

Threat Analysis Using Fuzzy Set Theory..... 455

Jaime Cisneros, Clark R. Karr, Sumeet Rajput
Institute for Simulation and Training
Orlando, FL
Pamela McCauley-Bell
University of Central Florida, IE Dept.
Orlando, FL

Micro Resolution Terrain Processor (MRTP)..... 463

John A. O'Keefe

U.S. Army Natick RD&E Center

Natick, MA

Charles W. Howard, Paul Saucier

Raytheon Company

Tewksbury, MA

Control of a CGF Fireteam with Voice and Gesture Commands..... 471

Douglas A. Reece

Institute for Simulation and Training

Orlando, FL

Sensitizing Synthetic Forces to Suppression on the Virtual Battlefield..... 479

Michael L. Fineberg, Gene E. McClellan

Pacific-Sierra Research Corp.

Arlington, VA

Steven D. Peters

Micro Analysis & Design

Boulder, CO

Session 7b: Unit Control

Command Agent Technology in a War Game Simulation..... 493

Gary Preston

Logica, UK Ltd

London, NW1 2PL, England, UK

Janusz M. Adamson

Defence Evaluation Research Agency, Centre for Defence Analysis

Sevenoaks, Kent, England, UK

Representative Communications for the Purpose of Command and Control in

Computer Generated Forces..... 503

Jean Philippe Landry, S. Valade, Dave N. Siksik

CAE Electronics, Ltd

St. Laurent, Quebec, Canada

An Architecture for Integrating Command and Control Capabilities of Heterogeneous Simulations..... 511

Frederic McKenzie, Gregory Shumaker, Pete E. Campbell

Science Applications International Corp.

Orlando, FL

Drilling CGF Agents in METT-T: An Alternative Approach to Conventional AI..... 519
Richard W. Penney, M. J. Kirton
Defence Research Agency
Worcestershire, UK

Session 8a: Terrain Modeling

**Multiple Elevation Structures in the Improved Computer Generated Forces
Terrain Database..... 533**

Thomas Stanzione, Forrest Chamberlain, Larry Mabijs, Mike Sousa
TASC
Reading, MA
Alan B. Evans, Cedric B. Buettner, Jonathan Fisher, Howard Lu
Science Applications International Corp.
Burlington, MA

Representations of Buildings for Individual Combatant CGF..... 545
Douglas A. Reece, Hsiao-Kun Tu
Institute for Simulation and Training
Orlando, FL

**Ocean Representation in the Improved Computer Generated Forces Terrain
Database..... 555**

Thomas Stanzione, Forrest Chamberlain
TASC
Reading, MA
Alan B. Evans, Cedric B. Buettner
Science Applications International Corp.
Burlington, MA

**Global Coordinate System in the Improved Computer Generated Forces
Terrain Database..... 565**

Thomas Stanzione, Forrest Chamberlain
TASC
Reading, MA
Alan B. Evans, Cedric B. Buettner, Howard Lu
Science Applications International, Corp.
Burlington, MA

Session 8b: Advanced Concepts

An Adaptive Environment Modeling Method Under Uncertainty..... 573

Richard A. Alo', Moshen Beheshti, Andre de Korvin, Chenyi Hu, Ongard Sirisaengtaksin
Dept of Compt. & Math Sci., College of Sci. & Tech.
Houston, TX

Simulating a Battlefield Maneuver Using Reaction Diffusion Equations..... 583

MaryAnne Fields

US Army Research Lab

Aberdeen Prov Grd, MD

Flexible Teamwork for Intelligent Simulated Pilots..... 591

Milind Tambe

Information Sciences Institute, USC

Marina del Rey, CA

Distributed Modeling of Cooperative Behavior by Mobile Agents..... 599

Peter S. Sapaty

Dept. of Electronic & Electrical Engr., Univ. of Surrey

Guildford, Surrey GU2 5XH, UK

Authors List 614



Session 0: Plenary Session

Pratt, JSIMS

Next Generation Computer Generated Forces

David R. Pratt

Joint Simulation System Joint Program Office
12249 Science Dr. Suite 260, Orlando, FL 32826
prattd@stricom.army.mil

1. Abstract

Computer Generated Forces (CGF) originated with the advent of the computer wargames to support training and analysis. CGF really came into its own in the late eighties as part of the Simulation Networking (SIMNET) Semi-Automated Forces (SAF) program. Since then CGF have been an integral part of almost all Distributed Interactive Simulation (DIS) Exercises. The next major evolutionary step occurred when the Command Forces (CFOR) program incorporated Command and Control capabilities. The CGF community is now ready for the next major step forward. In this paper we will present a brief taxonomy of CGF and some of the inspirations and challenges for the next generation of CGF systems.

2. Disclaimer

This is an academic paper. The information and ideas are not part of - or used in the evaluation of the Joint Simulation System (JSIMS) Integration and Development Contractor Request For Proposal (RFP).

3. Introduction

As the computer matured and uses beyond straight number crunching were found, the military developed Computer Aided Training (CAT) war gaming. These first wargames were the automated equivalent of the "Risk" strategy game. This formed the basis of the constructive combat models in the years to come. More than anything else they kept track of the strength and location of the forces and resolved the engagements the users instigated. From these simple games, the roots of Computer Generated Forces emerged. The forces represented in these games tended to be aggregated and useful only for high level staff training due to the low resolution and fidelity of the models. The resolution and fidelity were limited by the available hardware resources and training audiences needs.

The major advance in CGF came about with the advent of the microprocessor. This provided the foundation for the development of the first virtual simulation system. It was quickly realized that it was not cost effective to build simulators for all the Battlefield

Operating Systems (BOS)¹. To round out the simulator units and add the Opposing Force (OPFOR), the Semi-Automated Forces (SAF) systems were developed. Currently, SAF represents the state of the art in CGF. However, we are starting to see pushes toward the next major evolutionary generation of CGF, the Autonomous Forces.

It is important to realize some of the tradeoffs that must be accomplished when developing a system. As shown in Figure 1, the developer and, in turn, the user must exercise tradeoffs between four important parameters. Resources - the number and type of people and equipment. Fidelity - How accurate are the models used to represent the cognitive and physical processes. Resolution - What level (platform, unit, etc.) is the battle space modeled. Execution Time - Speed of the model and execution of the model (faster / slower than real time). The determination of these tradeoffs is dictated by the exercise. It is here where the generations of CGF fit in, each has a niche on the overall spectrum of CGF possibilities.

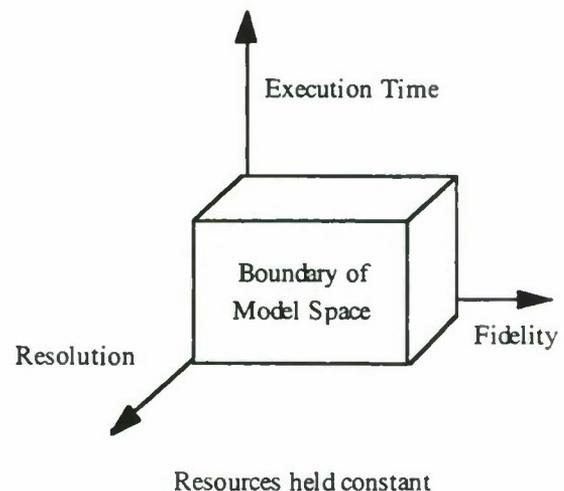


Figure 1: Tradeoffs in the Model Space

We have organized this paper into roughly two parts. In the first part we take a look back and develop a taxonomy for CGF systems. The second part looks

¹ More commonly called entities, but are sometimes represented as functional units.

to the future and tries to characterize what the next generation of CGF is going to be like. It is important to note that when we look into the future we see what might be, not what will be.

4. Generations of CGF

Like most taxonomies the categories can be partitioned in various ways and called different things. For the sake of this paper, we have chosen to develop a taxonomy based upon the cognitive processes used in the CGF systems. While this does not lend itself to a strict hierarchical breakdown, the generations can be seen in Table 1. This allows us to talk about general characteristics of each of the generations. One of the reasons why this is not a strict generational system is the current generation has not replaced the previous ones. This is due to the large number of requirements that the CGF systems fill. As a result, the generations tend to fill a niche requirement even after they have been supplanted technically.

Generation	Cognitive Process
1	None
2	Detection and engagement
3	Task interruption and execution
4	Multilevel Command and Control
5	Goal Selection and Learning

Table 1. The CGF Taxonomy

4.1 First Generation

The first generation of CGF is characterized by their complete lack of any cognitive process. As such, there is no deviation from the script that is laid down for them. While this might seem like a useless generation of entities, in reality they are some of the most instrumental and common CGF Systems. The data logger playback and traffic generator systems are typical of this generation. The data logger allows a prerecorded exercise to be repeated, observed, and analyzed over and over again. This provides the basic functionality for an After Action Review system. The exercise is recorded as it occurs, then it is analyzed after it is over to provide insight into what happened. More complex scenarios can be constructed by laying down new track over the existing ones or splicing pieces of existing tracks together. Likewise, the very definitive tracks are used in analytic evaluations since the paths and timing can be held constant over a large number of runs.

The noise generation systems are used to populate the battle field with a large number of low fidelity entities. The use of such a system is the track generator for a J-STARS like system, or to test a network's connectivity and throughput. At the Naval

Postgraduate School, we used a noise generator extensively when giving demonstrations to kids. The entities provided ample simple targets for them to shoot at.

The advantage to this generation of the CGF is that they are simple to use and require a minimum of resources to run. This allows a large number of entities to be placed on the battlefield with a minimum of resources.

4.2 Second Generation

The second generation of CGF added the ability for the entities to execute simple reactive behaviors that do not interfere with the planned actions. These behaviors are normally limited to detection, targeting, and engagement of hostile forces. The paths and routes are laid out by a user before or during the exercise; the reactive actions occur along these paths. Typical of these actions are for a unit to follow the route at all costs; the unit will charge on despite all of its peers getting killed, until it is killed, or until it reaches the end of the predetermined path. The complete lack of behavior reprogramming requires that the user pay close attention to the entities to ensure that they do not do something counter to common sense and doctrine.

The advantages of the second generation CGF are exemplified in the majority of current constructive models. These models are used for both training and analytical purposes. They tend to be manpower intensive to set up and run, but they are predictable. Some of them, particularly those used for analytical purposes, have a batch capability that allow multiple runs without operator interaction to determine the solution space for a given set of parameters. The rigidity of the behaviors limits the number of variables and aids in the analytic process.

4.3 Third Generation

With the advent of the virtual training systems came the realization that the aggregate Second Generation CGF systems could not adequately portray the individual entities on the battlefield. To satisfy this need the next generation of CGF was developed. This generation, commonly called Semi-Automated Forces (SAF), are the result of this work. These systems are typically a collection of preprogrammed tasks. The tasks themselves are made up of either a rule-based or state machine-based systems. What makes these simple behaviors so powerful is the nesting of the behaviors into task frames. The task frames are in turn nested within other task frames.

Missions are made up of a series of task hierarchies. This was a major advancement in the state of the art.

The hierarchy of tasks allows for complex reactive behaviors by creating a task queue. While not able to do goal selection, the second generation of CGF is capable of building complex missions with complicated reactive behaviors. The creation of the task frame sequences allows the user to create complex missions for the entities and, in some cases, simple units. More complex unit relationships can be built by linking the frames together in parallel.

4.4 Fourth Generation

The Third Generation CGF was a great step forward in the emulation of platform level entities. What is lacking is a representation of the Command and Control (C2) process. At its most simplistic level, the C2 process can be broken down in order to decide what the unit and subordinate units are suppose to do, and ensure it is carried out. While seemingly simple, it is one of the hardest tasks on the battlefield. The defining characteristic of the Fourth Generation CGF is the ability to replicate this process.

While there are many ways to insert C2 into an exercise, we shall limit our discussion to systems that represent the Headquarters unit in software. Even with this limitation there is a further limitation that we will make which is to ignore the majority, if not all, of the staff functions outside of the combat operations. This is a very valid assumption since very few of the CGF systems support any of the staff functions. As a result, only the commander is represented. This reduces the problem space to having to model the decision, or cognitive, processes, and the communication, or information gathering and order dissemination, process.

Typical of the Fourth Generation of CGF is DARPA's Command Forces (CFOR) program. CFOR models the C2 process by representing it as a series of interactions and behaviors of command entities. This results in the C2 process being primarily an information flow process among command entities problems. To address this, the Command and Control Simulation Interface Language (CCSIL) was created to represent the information exchanges between commanders. CCSIL messages are then passed through the command's communication structure to emulate the real battlefield information flow. This limits the information to those commanders who would really have it.

Even with a realistic information content and flow, the decision making is done at the individual command entities. These are the originator and recipient of the data. To simplify the generation of the orders and taskings from the user developed behaviors, CFOR uses a layered architecture where the developers only concern themselves with the top layer, the

Command Entity Application. The developer interacts with the lower layers by means of a well defined API. Once the orders are created and distributed they are interpreted by a Third Generation CGF system. In the case of the CFOR, Modular Semi-Automated Forces (ModSAF) executes the orders.

5. Where are we now?

Roughly equal numbers of people cheer the success of the various CGF programs and deride them. The reason for this is quite simple, the systems are suffering from their success. By this, we mean that the highly successful programs are being put in a situation to do something that they were never designed to do, and are being criticized because they cannot perform the tasks with ease. Starting at the entity level, the state of CGF mirrors the current state of Distributed Interactive Simulation (DIS), both work well at the company and below level. Once the entity counts and command structures start getting above that, they start to break down. From the aggregate level, it is the opposite. The higher level units can be reasonably portrayed, but the individual platforms have problems. To compensate for these problems we have seen a series of aggregate level models interfacing with the platform level models with varying degrees of success. This is not necessarily a limitation on part of the systems' developers, rather it is a combination of modeling paradigms, resource limitations and funding, and research and development profiles.

One of the major problems with the government funding system is that it is much easier to incrementally add a capability to a system than it is to re-engineer it. This results in systems that are large and monolithic, since the funding agency "just wanted to add one feature, not redesign the system." Due to the growth pattern and architectural age, the current CGF system have become resource intensive since almost, if not all, of the capabilities are in every version of the system.

One of the major problems of a system architecture having a long life span is the clean interfaces and modular nature of the first version erodes as features are added. This makes it very difficult to find the core features of the system and as a result the system becomes hard to maintain and adapt.

Those who have gotten us here have accomplished a Herculean task. In doing so, they have overcome a large number of hurdles. However, if we are to satisfy the customers who have grown to expect miracles, it is time to bring on the next generation.

6. Challenges for the Next Generations of CGF

Given all that it has taken to get to where we are now, there is as much, if not more, to go before there is a CGF system that can repeatedly pass a Turing test. With that in mind, the remainder of this paper deals with some of the critical technologies that will have to define the next generation of the CGF.

6.1 Changing and Adding Behaviors

The true value of a CGF is the behaviors that are part of the model. Likewise, these are some of the hardest things to model. The reason for this is quite simple, it is very hard to express any cognitive process in a clear, unambiguous manner. Given that is the case, we are presented with the first of the major challenges - standardization and codification of processes and cognitive models. There are currently several tasks under way by the Joint Staff and the various services to do this. The outcome of these efforts could then be merged and encoded into a common conceptual model. It is this encoding, expressed in terms that the operators can understand and agree to, that could then be compiled to generate the behaviors of the entities on the battlefield. By compiling the behaviors straight from the operators' task list to a runtime format, we can save significant time and resources in the generation, modification, and validation of the behaviors that make up the model.

The development of a common compatible specification language resolves some of the needs of the behavior generation of the next generation of CGF; however, it does not solve all the problems. The user of the CGF systems needs to be able to generate new behaviors to represent the unique training objectives of the particular exercise. The behaviors and taskings need to be tunable to represent the human conditions. If the CGF entity has been in combat for the last twenty-four hours, the decision cycle is going to be a little longer and they might not be as aggressive.

6.2 Reduce Required Resources

With a few exceptions, the CGF have been developed for use by dedicated operators or gaming cells. This is roughly equivalent of fighting the war through an interpreter. The next generation CGF system will interact directly with the war fighter using their organic systems. This is a fairly broad statement that most people interpret to mean that the CGF will be controlled by the Command, Control, Communication, Computer, and Intelligence (C4I) systems. That is a part of it, but there are many more means of communication that are used. To explore the new interface paradigms, the CGF developers are going to

have to interact to new communities. For example, the use of speech as both an input and output mechanism is starting to reach a point of maturity where it is robust enough to be useful in a fielded system. This opens up the possibilities of the synthetic radio network where the software scout can send a spot report back to the human commander and the commander can give him a new tasking. In order for this to happen, natural language processing will have to evolve to a point where the messages can be parsed and understood with some degree of reliability. The new interfaces are not limited to speech; gestures and image understanding also play a part. In a field exercise, the commander or the operations staff will mark up a map as they develop the plan. After this, an operations order is developed and briefed. By understanding the meaning of the overlays, the text of the order, and the gestures used in the briefing, the basis of the CGF operations of the exercise has been created. In small unit operations, the use of formal gestures, such as hand and arm signals, can represent a majority of the communication bandwidth between entities. Since one of the major functions of the CGF is to flesh out units, they should be able to take direction in the same manner as their real life counterparts.

The reduction in resources is not just in the set up on an exercise, the next generation runtime system has fundamentally changed from the current monolithic systems. The new systems take full advantage of the network computing paradigm that allows the processing of the data to migrate from one processor to another. The user's concern with the CGF is primarily - "is it doing what I want it to", not - "what is the CGF computing model and where are process executing." This only becomes a concern when the CGF is not providing the user with the responses in a timely and realistic manner. Taking advantage of this, the next generation CGF is based on the paradigm that there are services that are available on the simulation network, so use them. The existence of processing modules allows the system to dynamically alter where computations are done. The ability to do load shedding and balancing is central to the systems ability to reduce the number and power of machines required for the system to operate efficiently. By segmenting the CGF tasks into functional modules, they can be optimized and parallelized to increase the flexibility and scalability of the system. This way if there are no ships in the scenario, that capability will not have to have resources allocated to it even though they will not be used. This takes the "Dial a War" concept used in DIS down to the functional level.

6.3 Model Forces at a User Selected Level Of Resolution / Fidelity

The current state of CGF lends itself to the large monolithic systems geared to a particular level of forces. The next generation system will be built much more along the lines of the layered system shown in Figure 2. The foundation of these systems will be a common set of core Support Services. These are the parts of a CGF system that are introduced as simulation artifacts rather than models of real life processes. This includes such modules as the computer communication network interface (i.e. Run Time Infrastructure (RTI) interface), the process scheduler, and persistent object storage and management. This foundation is the most universal of the three layers and, as such, the most reusable.

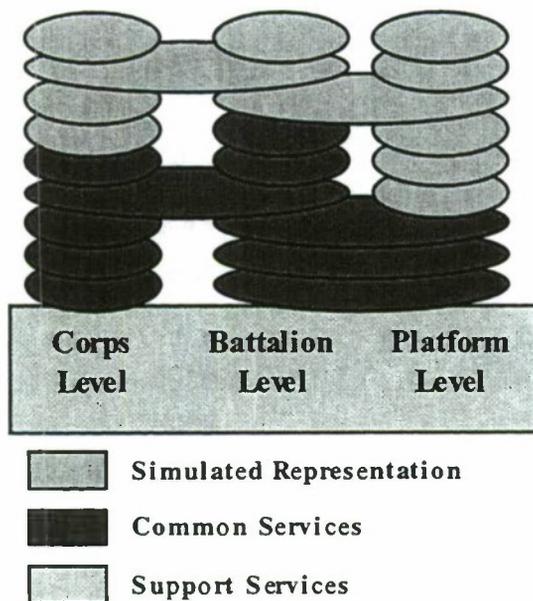


Figure 2: Interactions of Modules in a multifidelity CGF System

The remaining two layers of the next generation CGF system represent where the differentiation between CGF systems exists. At these layers, the Common Services and the Simulated Representation, the developer has to make the tradeoffs shown in Figure 1. This leads to differing implementations of the same military entity. For example, if a CGF system needs to run on a single workstation much faster than real-time, it will have lower fidelity and resolution than one that runs across a network of machines in real-time. However, if both systems had a consistent interface to the object, it would then be possible to replace one object with the other. This, in turn, gives the user the ability to select the object that they need for a particular exercise from a repository.

The next logical step is to have a single object that has multiple fidelities and resolutions internal to it. This would allow the ability of an object to be consistent within itself regardless of the echelon it is operating at. Using the terrain as an example, a plane is flying high overhead, it can see a large area of terrain, but at a fairly low level of resolution. As the plane comes in for a close air support mission the terrain changes resolution to match the fidelity needed for the ground targets to operate in. As the plane rolls out, the terrain is relaxed once again to allow the large area visualization. While the scenario above is done easily with level of detailing on a single station visualization system, it is much harder to do in a dynamic multiplayer system and in a system where the CGF has to reason about the terrain.

The middle layer of Figure 2, Common Services, is where the echelon modeling starts to make a difference in the types and the functionality of the modules in the simulation system. At this level the common services are those modules that help establish the common operating environment for the CGF, or are modules that apply across a wide range of CGF systems. This layer is comprised of such modules as the Synthetic Environment, mobility models, Line of Sight (LoS) processes, and the interconnections to the user's organic equipment.

The top layer contains the Simulated Representations, or physical and cognitive processes, of the CGF. As in the layer below it, the objects are represented by either a multifidelity object or a family of objects having the same interfaces. Once again, this allows the user to perform tradeoffs to compose the CGF mix that is appropriate for the exercise. However, at this level of abstraction and encapsulation the consistent interfaces also allows for the insertion of the human player at various echelons. The big advantage of the multi-resolution object representation is that the units are internally consistent with themselves. As a result the need for external aggregation / disaggregation no longer exists, since the object performs it internally.

6.4 Goal and Mission Selection

Perhaps, one of the greatest differences between humans and the rest of the animal kingdom is our ability to set goals, rationalize them, and make plans to achieve them. In order to reduce the number of controllers, the next generation CGF needs to have this capability as well. A goal, such as taking a hill, can be assigned to an object by internal or external forces. Externally, it can simply be told to take the hill. Internally, it has to rationalize the larger context before it decides that taking the hill is to its advantage. The reasoning process is the hard part. To determine if the hill should be taken, several questions need to

be answered and tradeoffs need to be done in the answer space. In many ways, this is what the battlefield commander does as a matter of course, set the goals of the unit in the context of the overall mission.

Once the goal has been set, the next step is to plot out a mission, or how the goal is going to be achieved. Once again tradeoffs will have to be done. For example, the variables of expected number of friendly / enemy / neutral casualties, amount of terrain covered, types of equipment needed and available, possibility of future actions, etc.. all have to be considered. The next generation of CGF will have to be able to make these types of determinations if we expect them to represent forces at different levels while reducing the amount of human controllers.

Perhaps the biggest challenge is the reprioritization of the goals and mission. Current Third Generation CGF Systems have the ability to interrupt what they are doing to respond to external stimuli, such as mine fields and air attacks. Once the stimuli induced event is over, the mine field breached or the planes fly off, the original mission resumes. What is lacking is the ability to reprogram the goal based upon what just happened. In the case of the air attack, the enemy now knows where the CGF units are, so surprise is lost. As a result, the mission parameters have changed and the tasking and goals need to be reevaluated, and possibly altered, in light of the new information.

6.5 Learning

The final of the characteristics of the next generation of CGF system that we are going to discuss, knowing that there are others, is the ability of the CGF to learn. If we take a look at the rationale for building the majority of the CGF programs, we see that they were used to support training. The training that has been done has been completely on the human side. At the end of the exercise, it is the same CGF as that which started the evolution. The CGF should be able to learn from the exercise as well. For example, if one of its units runs into a minefield and gets hit with artillery fire, it might be a coincidence. The second time it happened, the CGF should see a pattern developing. The third time the CGF hit a minefield, it should be expecting the artillery fire and react accordingly. The ability to find trends and exploit them is a characteristic of a good commander. Likewise, repetitive actions and tactics allows the enemy to predict what is going to happen next and react to it. As the CGF assume the role of a battlefield commander, it needs to learn how to fight the war as well.

7. Conclusion

In this paper, we have presented three key topics: (1) How we arrived at the current state of CGF; (2) The fact that there are niches for many different kinds of CGF and no one monolithic system can satisfy all needs; and (3) There is still a lot of work to be accomplished, but we are poised to take the next great step. The next generation will be one step closer to the objective CGF system that is capable of plotting goals, strategies to achieve them, taking advantage of the opponent's mistakes, and exhibiting those human traits that make us individuals. At this point in time we will have a true Autonomous Force.

8. Author's Biographies

Dr. David R. Pratt is serving as the first Technical Director of the Joint Simulation System (JSIMS) Joint Project Office in Orlando, Florida. He holds this position concurrently with an appointment as a tenure track faculty member at the Department of Computer Science, Naval Postgraduate School (NPS) in Monterey, California. Prior to joining the faculty at NPS, Dr. Pratt was a Data Processing Officer in the United States Marine Corps. He holds a Ph.D. and M.S. in Computer Science from NPS and a BSEE from Duke University. He has an extensive publication record with over thirty published articles covering a wide range of computer topics.

Session 1a: Command Forces Simulation

**Salisbury, The Mitre Corporation
Calder, SAIC
Goldman, Hughes Research Labs
Gratch, ISI/USC**



Command Forces (CFOR) Program Status Report

Susie M. Hartzog
NCCOSC RDT&E Division, code 44205
53140 Systems Street
San Diego, CA 92152-7560

Marnie R. Salisbury
The MITRE Corporation
1820 Dolley Madison Blvd
McLean, VA 22102

1. Abstract

The command forces (CFOR) program is implementing a new aspect of warfare simulation: explicit modeling of command and control. The program adds three major elements to the corpus of combat simulation: (1) an architecture where software simulation of command and control interacts with the simulated battlefield through a set of common services; (2) a common language for information between and among command entities and human participants; and (3) a development strategy that integrates the efforts of multiple developers to produce a functioning multi-service command forces simulation.

The CFOR program has passed through the concept and planning phases and is being implemented. This paper presents a brief overview of the three major elements along with a description of the current status of the program and its near term objectives.

2. Background

The Command Forces (CFOR) project is a part of the Synthetic Theater of War (STOW) program, an Advanced Concept Technology Demonstration (ACTD) that is jointly sponsored by the United States Atlantic Command (USACOM) and the Defense Advanced Research Projects Agency (DARPA). The STOW program is scheduled to support a USACOM exercise in 1997 where entities from each US armed service will interact with each other and with credible opposing force objects in a virtual simulation environment. The STOW ACTD will be the first large-scale demonstration of a High Level Architecture (HLA) simulation Federation supported by the HLA's Run Time Infrastructure.

The STOW ACTD requires the representation of larger-scale and more diversified military operations in virtual simulation. A key element in achieving this goal is the ability to represent both fighting forces and their commanders in software. CFOR extends the current entity level simulation architecture to incorpo-

rate explicit, virtual representation of command nodes, C2 information exchange, and command decision making.

3. CFOR History

The CFOR concept and program was born in the Fall of 1993 in response to DARPA's concern with the vertical scalability of entity-level simulations. DARPA's goal in the Advanced Distribution Simulation project was to provide a high resolution, high fidelity battlefield simulation that would support Joint Task Force level training. The modeling techniques being applied at that time did not seem likely to achieve realistic simulated behavior for larger and more complex force structures. After studying the problem we determined that the vertical scalability problem might be solved by focusing on the command and control entities that synchronize and direct the activities of the forces applied in a battle. Our theory was that the basic actions of an individual tank or airplane are fairly straightforward. Complexity arises from the organization of platforms into units that can execute temporally and spatially sophisticated actions to accomplish goals. The key tasks of organizing platforms into units and directing and controlling their actions are accomplished in the real world by battlefield commanders.

4. CFOR Contributions

The CFOR program adds three major elements to the corpus of combat simulation. These three elements are described here.

4.1 CFOR Architecture

The CFOR architecture was devised to allow for experimentation in the application of cognitive modeling techniques to the problem of simulating battlefield commanders. The architecture is flexible in that it allows multiple developer teams to explore different technical approaches for developing sophisticated models of battlefield commanders and necessary de-

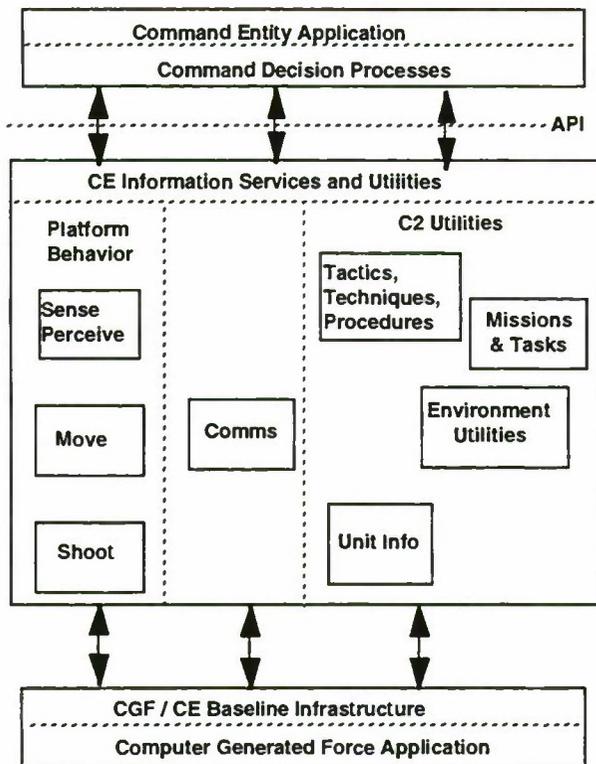


Figure 1: CFOR Technical Reference Model

cision-makers and to link those models to the existing entity-level simulations. In the CFOR program, these models of commanders and decision-makers are called Command Entities.

The CFOR architecture is portrayed best by the technical reference model (TRM). This TRM (see Figure 1) promotes interoperability and coherent C2 activity by providing a shared infrastructure, a common set of information and computing services, accessible through a well-defined applications interface.

The TRM is composed of three layers: Application Layer, Information Services and Utilities Layer, and Baseline Infrastructure Layer. This layered approach provides three specific benefits: 1) it provides a means of centralizing control over the baseline of doctrinal knowledge needed by the command entity applications; 2) it reduces command entity developers' efforts by providing common reusable software; and 3) it shelters the command entity developers from technology and functional enhancements in the baseline applications (e.g., ModSAF) and allows them to focus on command decision behavior.

- The Command Entity Application layer is where the command decision-making processes reside.

Command Entity Applications may be fully automated software or C2 workstations operated by human command entities. All details about the actual implementation of a software command entity are under the purview of the simulation developer organizations; they are free to implement their own approach to making command decisions. Likewise, the adaptation of C2 workstations to the CFOR architecture is dependent only on the interface specification to selected modules with the Information Services layer. Workstation developers are free to decide how to display, massage, or augment the simulation data available via the Information Services layer.

- The Information Services layer contains services and utilities that provide the information needed to support command decisions. These services impose few restrictions on how to model the decision process. They avoid making any inferences or judgments that are the proper purview of command entities.

Access to the services and utilities is specified by an Application Programmer's Interface (API) written in Interface Definition Language (IDL).

Services available include the following:

Platform Behaviors provide a generic interface to a command entity's physical representation on the battlefield. A command entity is associated with a vehicle or a set of vehicles (e.g., a command post). For example, an Army Company commander may ride in a tank, a Bradley Fighting Vehicle, a helicopter, or a HMMWV. Services provided mimic the commander's ability to sense from his vehicle, move his vehicle around the battlefield, and employ his weapons. In the past two years the platform behaviors have been extended as the underlying application responsible for modeling the commander's vehicle has become more capable. For example, a command entity can now request information about the atmospheric conditions observable or discernible from his vehicle.

Communications offer an application interface to Command and Control Simulation Interface Language (CCSIL) message utilities. (see below for a discussion of CCSIL)

C2 Utilities represent the background knowledge and rote reasoning capability of the commander—"routine" knowledge, shared by every human com-

mander, that does not depend on subjective judgments. This is important for several reasons:

- To prevent redundant and potentially inconsistent knowledge acquisition and engineering efforts by the command entity developers.
- To help focus the activities of the command entity developers on addressing the difficult issues in modeling subjective, context-sensitive judgments and decisions.
- To localize the encoding of doctrinal information within the CFOR family of application software for two reasons: 1) to facilitate CFOR testing and evaluation; and 2) to minimize the effort needed for future enhancements or modifications for particular exercises or scenarios.

Services include

Environmental Utilities which provide the ability to compute mobility corridors, control measures, reverse slopes, routes, travel time and speed. (Environment includes terrain, ocean, and atmosphere.)

Unit Info which provides access to static data about units (own and enemy) and the ability to make basic inferences (e.g., combat power) from the raw data.

Missions and Tasks which provides doctrinal decision templates to help interpret an ordered mission and to devise a plan.

Tactics, Techniques, Procedures which provides templates to help fill out orders and implement a plan.

- The Baseline Infrastructure Layer contains the basic platform representation and general DIS interface utilities. These capabilities are accessed by command entity applications indirectly through the Information Services layer. For the STOW ACTD the baseline infrastructure layer includes the four Synthetic Force applications: Army SF, Navy SF, MC SAF, and AFSAF.

4.2 CCSIL

The Command and Control Simulation Interface Language (CCSIL) is a special language for communicating between and among command entities and small units of virtual platforms generated by computers for the STOW ACTD environment. CCSIL includes a set of messages and a vocabulary of military terms to fill out those messages. It was developed to facilitate interoperability between different implementations of command entities and platform entities (vehicles) in an HLA Federation.

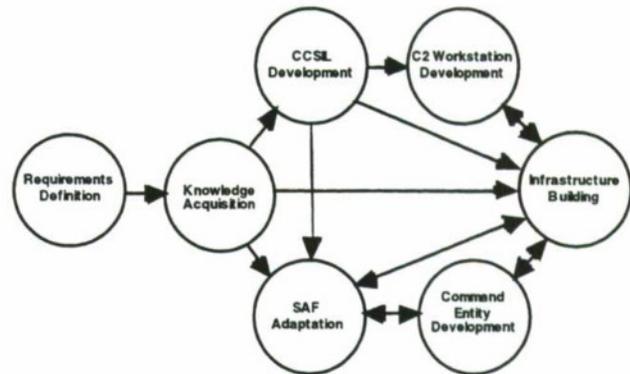


Figure 2: Activity Relationships

A common language designed for interpretation by software is needed to allow all three implementation approaches (workstation, automated command entity, and SAF) to work together in one environment. By using the structured format of CCSIL messages, humans at real world command and control workstations can send orders and directives to software command entities and expect them to react appropriately. Likewise, software command entities can exchange messages with each other.

Without a common language and communications services, every new element added to a Federation would need to be iteratively retrofitted to interoperate with every other existing element of the virtual simulation Federation. CCSIL serves as a unifying thread among diverse implementations of command entities, computer generated forces, and command and control workstations.

4.3 CFOR Development Process

The process for developing a fully operational CFOR system is depicted in Figure 2 and described in the following paragraphs.

The process is being applied to each of the Services independently, although oversight over the entire program is being applied by the program System Engineer.

- *Requirements Definition.* The first step in implementing CFOR is deciding and documenting which C2 elements will be represented in simulation, which missions they should perform, and how each of them will be implemented (human, automated

commander, or SAF). This concept is developed in close coordination with Service representatives.

- *Knowledge Acquisition.* Experts in each field and for each military Service gather information about the command process. Particular emphasis is placed on planning, decision-making, monitoring, and revising plans. After initial gathering and documenting by contractors, the Services will assume responsibility for maintenance of the knowledge base.
- *CCSIL Development.* CCSIL is based on the product of knowledge acquisition—on the documented C2 process and the identity, format, and content of relevant message exchanges. The CCSIL development team works closely with the knowledge acquisition team to assure clarity and completeness.
- *C2 Workstation Adaptation.* Selected C4I Systems will be integrated with the Modular Reconfigurable C4I Interface (MRCI) to enable the warfighter to participate in the virtual simulation via their real world system. To the extent possible, the MRCI project is adopting the CCSIL message set as a starting point for defining the standard for information exchange between real world C4I devices and simulations.
- *SAF Adaptation.* ModSAF is being enhanced to model new vehicles and small units and to model new behaviors for entities and small units. This version of ModSAF is then adapted to properly carry out CCSIL orders and requests and to generate CCSIL reports. The CCSIL adaptation has been integrated into the ModSAF 2.1 baseline.
- *Command Entity Development.* The CFOR program plan calls for multiple contractors, each developing a software implementation of a command entity. For each command entity, the contractor builds the required mission behaviors. After a suitable period of development, the implementations are evaluated. Subsequently, the developers continue to deliver additional mission areas and new command entities on an approximate schedule of every three months until the 1997 demonstration.
- *Infrastructure Building.* The CFOR infrastructure software provides services to the command entity simulation and the real world C2 systems based on information provided by the knowledge acquisition process. An initial delivery of this software was made in January 1995; new versions are issued

every one to three months, accommodating new CCSIL messages and modifications needed by Command Entity developers.

- *Testing and Integration.* The nature of the CFOR program dictates steps beyond the normal testing process. Technical integration testing is needed to assure that all components communicate correctly. Also Command entity behavior must be evaluated against reasonable behavior standards, initially by the knowledge acquisition teams and ultimately by Service experts.

5. CFOR Development Status

The majority of the CFOR development work accomplished to date has been in the Army domain. However, some work has been completed for the other military Services. Using the general outline described in Section 3 for the CFOR Development Process, the following paragraphs briefly describe the status of the CFOR effort.

5.1 Army

Army requirements definition started in October 1993. Based on the requirement for the Army to simulate a heavy brigade as part of the Joint Task Force for STOW 97, we determined that the initial command entity to be developed would be an Armor/Mech Company Team Commander. Additional command entities to be developed include the Company FIST, the Company Trains Commander, the Engineer Platoon Leader, and the Battalion Commander. In order to support ground maneuver operations for a Mech Heavy or Armor Heavy Brigade Task Force, the mission areas being developed are Attack, Defend (including defense in sector, defense of a battle position, and reserve unit in the defense), and Movement to Contact. The overall goal is to provide a combined arms capability with emphasis on maneuver and fire support. To address Rotary Wing Air (RWA) requirements, RWA Company and Battalion Commanders are being developed that will be capable of performing Attack, Reconnaissance, and Security missions.

Logicon RDA has the responsibility of providing Army CFOR knowledge acquisition (KA). Based on the above requirements, Logicon's approach has been to identify key elements in the decision process based on Army doctrine. In particular, the KA team has used the Army Training Evaluation Program (ARTEP) collective tasks with particular attention to

C2. These individual ARTEP tasks combined with descriptions of higher order decision making to collectively provide the basis of the knowledge for the command entity development.

The current CCSIL Ground Operations message set consists of about 39 messages that cover Orders and Directives, Unit Situation and Status Reports, Fire Support Messages, Engineer Messages, Air Defense Messages, and Combat Service Support Messages. Currently, the majority of messages being used fall into the Orders and Directives and Unit Situation and Status Reports categories.

Science Applications International Corporation (SAIC) has the responsibility for developing Army Ground Maneuver command entities. Initial development started with the Company Team Commander in January 1995. SAIC demonstrated the Company Team Commander performing an Attack mission in the STOW Engineering Demonstration 1 in October 1995. Since then, their focus has been on the Defend mission area, enhancing the Attack mission area, and on developing command entity to command entity communications so that eventually all companies will be able to operate and communicate effectively as part of a battalion. SAIC is also building the FIST command entity along with the interactions (guidance) that occur between the Company Team Commander and the FIST. SAIC initiated the Battalion Commander effort in May 1996.

The SAIC team's approach to automated decision making is based upon a Constraint Satisfaction Tool. Planning and replanning is performed by a Combinatorial Constraint Satisfaction (CCS) procedure which acts as an interpreter for high-level behaviors expressed as Constraint Sets (CS). Execution and monitoring is performed by Autonomous Control Logic (ACL+).

Information Sciences Institute (ISI) has the responsibility for developing RWA commanders. The initial effort to develop a RWA Company Commander capable of performing an Attack mission is well underway. Using the SOAR technology, ISI has built an RWA Company Commander that can plan for and direct a force of RWA pilots also built in SOAR.

The Army CFOR testing methodology has been to test the command entities in several virtual Situational Test Exercises (vSTXs) and virtual Field Training Exercises (vFTXs) which, collectively, make up the unit level testing of the command enti-

ties. The main purpose of the vSTXs and vFTXs is to assess the reasonableness of behaviors within the ARTEP construct.

SAIC's and ISI's Army command entities will be further tested in STOW's Combined Behaviors Test 1 in July 1996. The Army portion of this multi-Service test will occur at the National Simulation Center (NSC) at Fort Leavenworth, Kansas. Army Service experts will be present to evaluate behaviors.

5.2 Navy

Navy CFOR requirements definition started in February 1995 and continues. The Navy CFOR effort has focused on Navy CCSIL development in support of both sea and air operations.

The current CCSIL Sea Operations message set consists of about a dozen messages that cover Sea Mission Control, Anti-Air Warfare (AAW), and Anti-Surface Warfare (ASuW) components of the Navy's mission space. Additionally, Link 11, OTH Gold, and ATP-1 message sets have been identified. The Navy Synthetic Force entity development team has adapted their simulation to send and receive CCSIL messages and has developed an initial implementation of Link 11.

The current CCSIL Air Operations message set, which supports both Navy and Air Force air operations, consists of about 45 messages that cover Close Air Support (CAS) Mission Control, Brevity Codes, and Air Mission Control.

BMH Associates has the responsibility of providing Navy KA. To provide a CCSIL capability within the Navy, BMH has developed two storyboards that will closely tie in sea assets being represented by Navy SF and air assets being represented by Soar Fixed Wing Air Intelligent Forces (IFOR). These two storyboards will demonstrate Close Air Support (CAS) and Anti-Air Warfare (AAW) missions.

The Link 11 message as well as the CAS storyboard and its supporting messages will be tested in STOW's Combined Behaviors Test 1 in July 1996. The Naval sea component will be tested from NRaD in San Diego, California and the Air component will be tested from the WISSARD lab at NAS Oceana, Virginia.

5.3 Marine Corps

Marine Corps CFOR requirements definition started in February 1995 and continues. Because a major thrust of the Marine Corps Synthetic Force program is to develop the Individual Combatant, we decided that the first command entities to be developed would be an Infantry Platoon Commander and an Infantry Company Commander. To support the Marine Corps role in STOW 97 in the areas of ground maneuver and amphibious operations, the mission areas being developed are attack, link-up, movement to contact, and hasty defend. The overall goal is to provide a combined arms capability with emphasis on being able to flexibly task organize Marine Corps assets into the force packages necessary for the mission.

BMH has the responsibility of providing Marine Corps knowledge acquisition (KA). BMH's approach has been to provide KA based upon Marine Corps doctrine. In particular, the Mission Performance Standards (MPS) from the Marine Corps Combat Readiness Evaluation System (MCCRES) and the Battle Drills established by Marine Corps Order have been collectively used to provide a framework to guide development and testing.

To support Dismounted Infantry, several of the messages in the CCSIL Ground Operations message set were enhanced. New CCSIL tasks and enumerations were provided to support Marine Corps lifeforms and munitions. Currently, the majority of message types being used fall into the Orders and Directives and Unit and Status Reports categories.

Hughes Research Laboratories (HRL) has the responsibility for developing Marine Corps Infantry Platoon and Company Commanders. HRL's initial effort was in developing an Army Company Team Commander in 1995. This was done simultaneously with SAIC's Army Company Team Commander effort in order to mitigate risk. HRL demonstrated the Company Team Commander performing an Attack mission in December 1995. Since January 1996, HRL has been entirely focused on developing the Marine Corps Infantry Platoon Commander. An attack mission capability will be provided first. HRL will soon begin developing the platoon commander to platoon commander interactions and communications so that the Company command entity can be realized.

The HRL team calls their implementation the Canonical Commander Model (CCM). The CCM comprises several distinct modules: a mission ana-

lyzer/planner, a friendly and enemy situation analyzer, and a terrain analyzer. A major technology component of the CCM is an inference engine that works over a set of fuzzy logic tables containing specific military decision making knowledge.

The unit level testing for the Marine Corps command entities will follow the methodology used in the Army program, namely to run through several virtual Situational Test Exercises (vSTXs) and virtual Field Training Exercises (vFTXs) where reasonableness of behaviors will be assessed in accordance with Mission Performance Standards (MPS) and the Battle Drills framework.

HRL's Marine Corps command entities will be further tested in STOW's Combined Behaviors Test 1 in July 1996. The Marine Corps component of this test will occur at NRaD in San Diego, CA. In the future the Marine Corps CFOR work will be integrated into the LeatherNet facility which is being used for training and mission rehearsal at 29 Palms, CA.

5.4 Air Force

Air Force CFOR requirements definition started in December 1994. The initial concept was to build an Airborne Control Element (ACE). However, this has been superseded by a requirement to develop an automated Wing Operations Center (aWOC). The automated WOC will receive an Air Tasking Order (ATO) in CCSIL and generate most of the necessary data to launch simulated aircraft on missions. This data will be forwarded to the Soar exercise editor and stored in a database accessed by the Soar FWA pilot entities. We expect that the aWOC will have a limited capability and that a human will be required to provide the detailed routing information needed to execute a mission. However, this initial capability will greatly ease the burden of the STOW operators in sortie generation. This effort is expected to start in July 1996.

Many of the existing CCSIL Air Operations messages will be reused to support the exchange of C2 information between Soar FWA pilots and other command decision makers that may be represented in software or played by humans, such as Forward Air Controllers.

6. Summary

CFOR is implementing explicit modeling of command and control by adding three major elements to

combat simulation: (1) an architecture where simulation of command and control interacts through a set of common services; (2) a common language for information among command entities and human participants; and (3) a development strategy to integrate the efforts of multiple developers to produce a multi-service command forces simulation.

CFOR has completed the concept and planning phases and is being implemented. This paper presented an overview of the three CFOR elements and a description of the status of the program and its near term objectives.

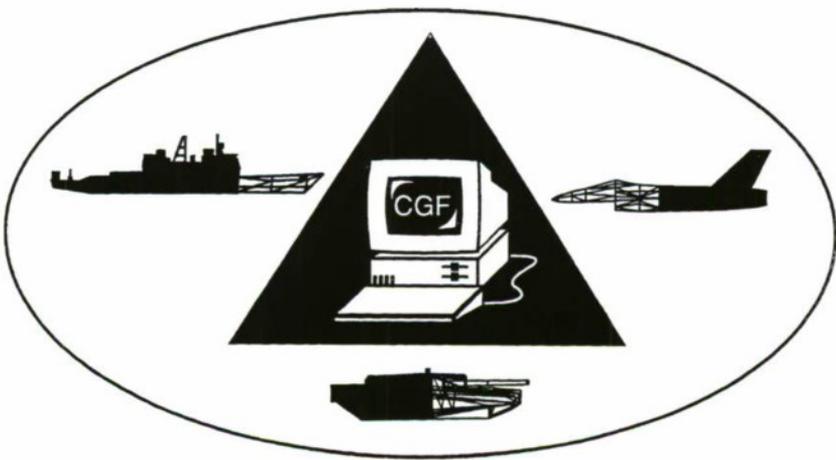
7. References

- Dahmann, J. S., M. R. Salisbury, L. B. Booker, and D. W. Seidel. 1994. Command forces: An extension of DIS virtual simulation. In *Proceedings of the Eleventh Workshop on Standards for the Interoperability of Defense Simulations*, 113-117. Orlando, Florida.
- MITRE Corporation. 1995. *Command and control simulation interface language (CCSIL) message content definitions*, McLean Virginia.
- MITRE Corporation. 1995. *Command and control simulation interface language (CCSIL) usage and guidance*, McLean Virginia.
- MITRE Corporation. 1995. *Command forces (CFOR) environment utilities application programmer's interface (API)*, McLean Virginia.
- MITRE Corporation. 1995. *Command forces (CFOR) infrastructure interface definition*, McLean Virginia.
- Salisbury, M. R. 1995. Command and control simulation interface language (CCSIL): status update. In *Proceedings of the Twelfth Workshop on Standards for the Interoperability of Defense Simulations*, 639-649. Orlando, Florida.
- Salisbury, M. R. , L. B. Booker, D. W. Seidel, and J. S. Dahmann. 1995. Implementation of command forces (CFOR) simulation. In *Proceedings of the Fifth Conference on Computer Generated Forces and Behavioral Representation*, 423-430. Orlando Florida.
- Salisbury, M. R. , L. B. Booker, D. W. Seidel. 1995. A Brief Review of the Command Forces (CFOR) Program. Presented at *The 1995 Winter Simulation Conference*, 3-6 December, Arlington VA.
- Seidel, D. W., M. R. Salisbury, L. B. Booker, and J. S. Dahmann. 1995. CFOR approach to simulation scalability. In *The Electronic Conference on Scalability in Training Simulation*. The Society for Computer Simulation, Institute for Operations Research and Management Science.

8. Authors' Biographies

Susie M. Hartzog is the Project Manager for the Command Forces (CFOR) Program at NRaD in San Diego, CA. She has a Bachelors Degree in Electrical Engineering from the University of California, San Diego. She has spent the last three years working in the area of Advanced Distributed Simulation (ADS). Ms. Hartzog is currently the head of the Advanced Behavioral Representation team at NRaD.

Marnie R. Salisbury is a Lead Simulation Engineer at the MITRE Corporation. For the past year, she has served as project leader for the CFOR team at MITRE. She is currently serving as the Technical Director for the STOW ACTD. Ms. Salisbury has ten years experience in military command and control and battle simulation.



Architecture of a Command Forces Command Entity

Robert B. Calder, Richard L. Carreiro, James N. Panagos, G. Robert Vrablik, Dr. Ben P. Wise
SAIC
Suite 130
20 Burlington Mall Road
Burlington, MA 01803
rcalder@bos.saic.com, rcarreiro@bos.saic.com, jpanagos@bos.saic.com, rvrablik@bos.saic.com,
bwise@bos.saic.com

Forrest L. Chamberlain, Douglas P. Glasson
TASC
55 Walkers Brook Drive
Reading, MA 01867
flchamberlain@tasc.com, dpglasson@tasc.com

1. Abstract

Representing command and control decision-making in software is a critical and challenging task confronting the simulation community. As the focus of Distributed Interactive Simulation shifts towards larger-scale, higher-fidelity exercises, there is an increased requirement for software implementations of intelligent command entities at higher-level military echelons. Current computer generated forces systems have achieved the reasonable simulation of individual platforms and small units. The Command Forces project endeavors to realistically model the complex command and control decision-making process of higher-level unit (i.e. company and above) commanders in the military hierarchy.

This paper presents the software architecture of a CFOR command entity which has been designed and implemented to achieve the goal of simulating this high-level decision-making behavior. The first application of this architecture is aimed at modeling the behavior of various Army commanders at the company and battalion levels. Descriptions of the key components of the system and details of the interactions which occur among these components are presented.

2. CFOR Overview

The Command Forces (CFOR) project is a part of the Synthetic Theater of War (STOW) program, an Advanced Concept Technology Demonstration (ACTD) that is jointly sponsored by the U.S.

Atlantic Command (USACOM) and the Defense Advanced Research Projects Agency (DARPA). The STOW ACTD is focused on training commanders at multiple levels up to the joint task force level, and therefore requires the ability to represent large-scale, diversified military operations in simulation. A key element in achieving this goal is the ability to represent both fighting forces and their commanders in software. Current computer generated forces (CGF) systems provide the simulation of individual platforms and small units. CFOR extends the basic DIS architecture to incorporate explicit, virtual representation of command nodes, command and control (C2) information exchange, and command decision-making.

The CFOR concept and technical reference model are described in full detail in [Salisbury, et al, 1995]. This paper focuses on the software architecture of a simulated commander, called a Command Entity (CE), which is capable of performing the planning, execution, tracking, and replanning of missions for various military commanders.

3. CE Application Areas

The CE architecture presented below has been designed to support the modeling of C2 decision-making for commanders at various echelon levels in multiple service areas. The initial application of this architecture has been the modeling of an Army Armor Company Team commander.

3.1 Army Armor Company Team Command Entity Capabilities

The Army CFOR program has devised a CE capability assessment model in terms of Mission, Enemy, Terrain, Troops, and Time Available (METT-T). This model is used to define the behavioral capabilities of the CFOR CE, and as a basis for determining test plans for the CE. The missions and behaviors which the CE can perform are based on published Army doctrine, and are traceable back to the Army Training and Evaluation Program (ARTEP) tasks defined for a given unit type.

Using the METT-T model, the capabilities of the Army Armor Company Team CE include:

Mission: The CE plans and executes offensive and defensive missions as part of a battalion task force. It plans and performs attack, defend, and reserve missions utilizing the appropriate company-level ARTEP tasks. It plans and performs explicit tasks which were specified in the battalion operations order, and also identifies, plans, and performs implicit tasks which were not specified by the battalion, but are required for successful execution of the mission. It properly handles a variety of unplanned events, such as encountering unexpected enemy ground or air units and encountering obstacles.

Enemy: The CE incorporates expected and actual enemy units and enemy force ratios into its planning and execution.

Terrain: The CE operates on open, desert terrain and rolling, wooded terrain. It incorporates expected visibility and mobility into its planning process.

Troops: The CE constructs plans for an Armor Company Team, which can consist of any mix of tank and mechanized infantry platoons, ranging from two to five platoons.

Time Available: The CE considers the time available to perform its mission during the planning process. This affects various factors of how the mission can be accomplished, such as route selection.

4. Command Entity Architecture

A high-level block diagram of the required components for a CFOR simulation is shown in Figure 1. Within the context of this architecture, the

CE software exists as a separate process which models the C2 decision-making of one or more simulated commanders.

In addition to the CE simulation process, Figure 1 shows a higher echelon commander (either human or simulated) which must be present to perform the role of the commander to which the CE is responsible. The role of this higher echelon commander may be filled by a human at a C2 workstation or another CFOR CE simulation process. Currently, this role is filled by a human operating a menu-driven C2 interface. The primary function of this interface is to allow the operator to send and receive messages on simulated radio networks.

Figure 1 also shows a small unit forces simulation which must be present to perform the simulation of the subordinate units and entities which the CE is commanding. The role of these units may be filled by any CGF system which fully supports the CFOR applications programmer interface (API). Currently, this role is filled by a modified version of the Modular Semi-Automated Forces (ModSAF) program, called Adapted ModSAF, which fully supports the CFOR API.

All communication between the CE, higher echelon commander, and small unit forces is via the Command and Control Simulation Interface Language (CCSIL). CCSIL includes a set of messages and a vocabulary of military terms for filling out those messages. The definition and implementation of the CCSIL message set allows different implementations of CE's, C2 workstations, and CGF systems to communicate via a common language. CCSIL messages are sent in DIS signal PDUs over simulated radio networks. Examples of Army CCSIL messages are the Operations Order, Fragmentary Order, Situation Report, and Status Report.

As shown in Figure 1, the CE application interfaces with the CFOR infrastructure utilities via direct function calls and Remote Procedure Calls (RPC). The C2 Utilities and Environmental Utilities are libraries which are linked directly into the CE application and are therefore invoked via direct function calls. The Communications and Platform Behavior Services are libraries which are linked into the Adapted ModSAF and are therefore invoked via RPC.

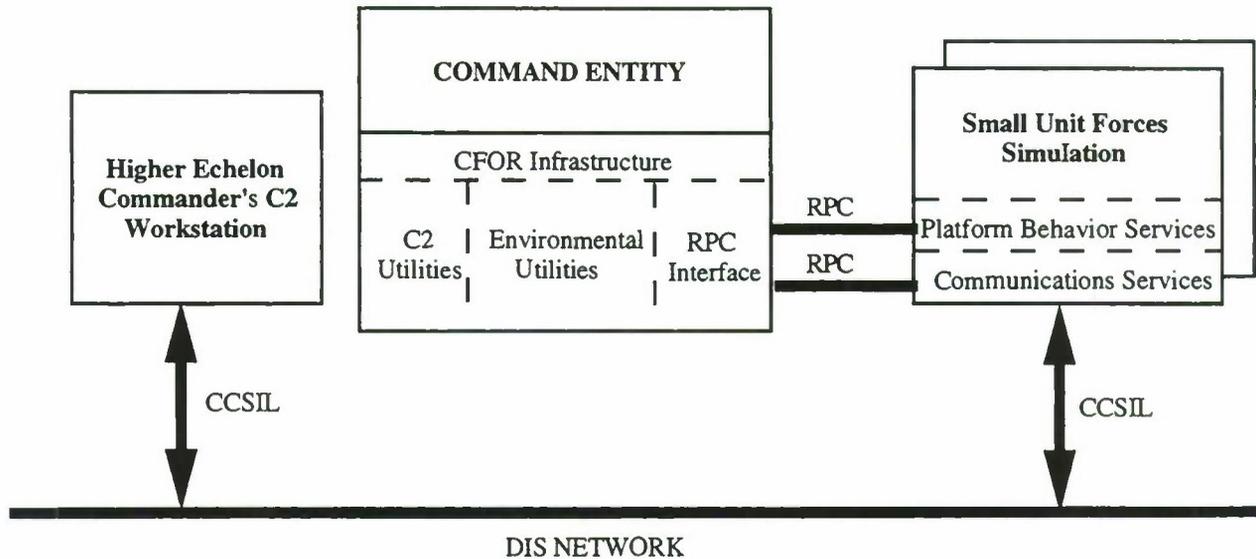


Figure 1: CFOR Simulation Component Block-Diagram

The architecture of the CE software has been designed to support the modeling of command and control decision-making for software commanders at multiple echelon levels in various service areas. It is organized such that general knowledge is contained in generic base classes and domain-specific knowledge is isolated in well-defined derived classes. This provides for maximum reuse of previously developed software, while not prohibiting implementations where specific knowledge is needed. The CE is designed utilizing an object-oriented methodology, and the software is implemented in C++. A high-level, object-oriented component diagram of the CE is shown in Figure 2. Figure 2 also shows critical data flows between the key components.

The following sections describe each of the major CE components in detail. These descriptions may have an Army bias, as that is the first application area to which this architecture has been applied, but the components presented are applicable to a CE in any service area.

4.1 Commander Class

A single CE process is capable of simulating multiple commanders simultaneously. The commanders simulated can be of similar or different echelons and roles. For example, multiple armor company team commanders, or a mix of armor

company team and fire support team commanders, can be simulated in a single process.

This capability is facilitated by encapsulating all of the components shown in Figure 2 inside of a commander class, and instantiating a separate commander object for each commander to be simulated. A non-interruptible, round-robin scheduling mechanism is used to give each commander object its slice of processor time. In order to ensure that each commander object gets its slice of the processor in a timely fashion, the planner class is constructed such that it returns control to the main scheduling loop if it utilizes the processor for more than a pre-specified amount of time. This is essential since the construction of an initial plan can take on the order of a few minutes. If the system allowed a single commander's planning to proceed uninterrupted for several minutes, it would cause all of the commanders being simulated to lose touch with the state of the simulated world. The approach implemented ensures that each commander, including the one which is performing the complex planning, will have timely access to events occurring in the simulated world.

The base commander class has derived classes for each of the different types of commanders which the CE application can simulate. Most components shown in Figure 2 also have similar derived classes.

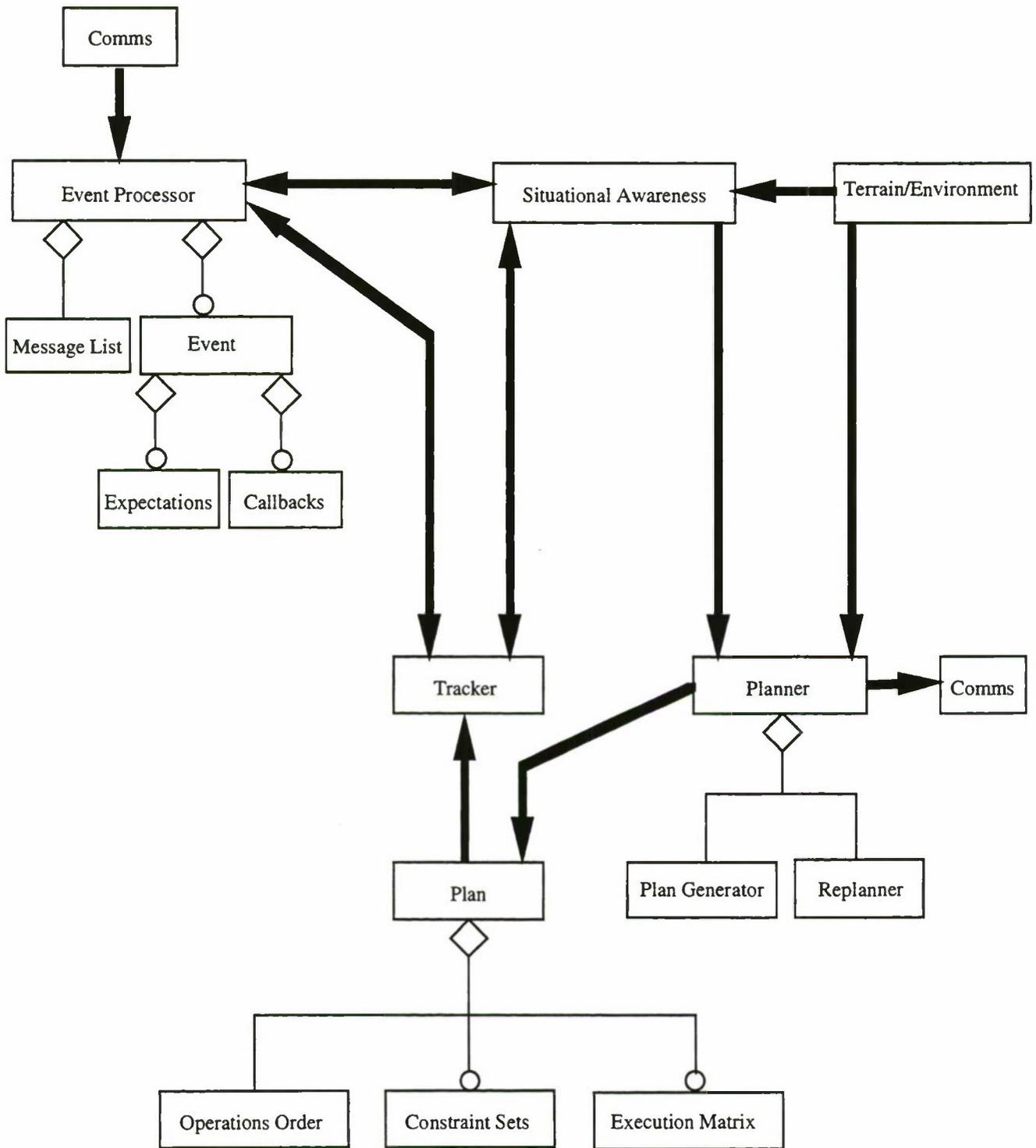


Figure 2: Command Entity Components and Data Flows

4.2 External Communications

The CE does not have a direct connection to the DIS network, and therefore does not read any DIS PDUs. Instead, it has two mechanisms for effecting changes in the simulated world and obtaining information about the entities and environment in the simulated world: 1) commands and queries of the platform, weapons, and sensors of the commander's vehicle, and 2) transmission and receipt of CCSIL messages. Both of these interfaces are implemented via RPC from the CE application to the CFOR infrastructure services.

The CE utilizes the Platform Behavior Service component of the CFOR infrastructure to interface to its own vehicle. The CE moves its vehicle, employs its weapons, and controls its sensors via this interface. This interface is also utilized by the CE to find out its location, speed, weapon status, and sensor status.

The CE communicates with other entities in the simulation via CCSIL messages. It receives CCSIL orders and intelligence messages from its higher echelon commander, and sends CCSIL situation and status reports to its higher echelon commander on a simulated radio network. It sends CCSIL orders to its subordinate units, and receives CCSIL situation and status reports from its subordinate units on a separate simulated radio network.

Each CCSIL message is a well-defined data structure. However, many CCSIL messages are complex data structures which contain sub-structures, optional fields, and variable length lists. In order to ease access to incoming CCSIL messages and construction of outgoing CCSIL messages, all of the CCSIL structures and messages have been encapsulated into C++ classes in the CE software. Each C++ CCSIL class has methods to access and set all of the CCSIL structures and fields within that CCSIL structure. Therefore, the CE components which access and manipulate CCSIL information never operate directly in the CCSIL message format. Instead these components access and manipulate the C++ CCSIL class objects. Each C++ CCSIL class has a method which converts an incoming CCSIL structure into its corresponding CCSIL C++ object. Additionally, each C++ CCSIL class has a method which converts it into its corresponding CCSIL structure. This encapsulation approach provides the

advantages of object-oriented programming for the entire CCSIL message set.

4.3 Event Processor

Each CE's execution thread is event-driven. The individual components of a command entity are responsible for identifying and registering all Events (discussed below) that are potentially relevant to their operation.

The Event Processor maintains a queue of all such events, and is responsible for identifying when any event on the queue has occurred, and triggering the desired response.

4.3.1 Events

Events are objects that define something that can occur in the system that requires that the system react in some way. They are implemented as instances of classes derived from the abstract Event base class. The Event base class defines a generic interface for each event, which includes three critical components: how to determine whether the event has occurred; what, if anything, is expected to be true about the state of the world when it occurs; and what to do when it has occurred.

Each subclass of Event defines its own Boolean "Occurred" function, which returns true when the conditions for the event have been met, and false when they have not. This function can then be queried by the Event Processor to determine when the event has occurred. For example, the "CommEV" event is considered to have occurred whenever a new CCSIL communication is received.

4.3.1.1 Expectations

Each event has associated with it a list of zero or more Expectations. Each expectation encodes the desired set of values for some characteristic of the world when an event occurs. For example, if the event is "Unit A crossed phase line Alpha," an expectation might be that this occurred before time T. Expectations, which are primarily used by the Tracker, are used to determine whether the mission is progressing according to the plan, based on a mission-specific set of parameters and tolerances.

4.3.1.2 Callbacks

Each event also contains a list of zero or more Callbacks. Each callback object encapsulates an action which should take place when the event occurs, such as informing the Tracker when a subordinate has crossed a phase line. As with events and expectations, callbacks are implemented using a base class which defines a generic interface, and subclasses which define actual functionality. The Event Processor is responsible for triggering each callback of each event that occurs. Note, however, that the functionality of each individual callback is completely hidden from the Event Processor.

4.3.2 CCSIL Message List

Incoming communications, representing orders and intelligence from the CE's superior, as well as status and situation reports from the CE's subordinates, account for many of the events handled within each CE. In order to efficiently handle this message traffic, the Event Processor is responsible for pulling messages off of the incoming communications queue, and storing them in a manner that renders them easily accessible by interested events.

4.4 Situational Awareness

In order for the CE to perform its mission planning, execution, and tracking, it must have a representation of its perception of the current state of the world. The Situational Awareness (SA) class provides this representation. In the Army context, C2 decision-making is performed based on the factors of Mission, Enemy, Terrain, Troops, and Time Available (METT-T). The SA class performs processing to build and store data regarding mission, enemy, troops, and time available. Due to the complexity of processing and the volume of data required for terrain processing, the CE architecture represents processing and knowledge of the terrain as a separate class, which is described below.

SA processes information from multiple sources, including CCSIL orders and intelligence messages from the higher echelon commander, CCSIL reports from subordinate units, and sensory data from queries via the platform behavior services. It uses this reported and sensed information directly, and also generates derived data from this information, to build the picture of the commander's view of the world.

SA registers with the Event Processor to receive all CCSIL messages which are sent on the commander's radio nets. As it receives each message, it extracts the information relevant to the commander and updates the corresponding SA data. SA also periodically queries the CFOR platform behavior services to obtain sensory information from the commander's vehicle and updates this data in the SA state.

A difficult problem encountered when building up state information from multiple sources is the proper handling of repeated or contradictory data in reports. SA handles this to some degree by performing simple fusing of data from multiple reporting sources, such as merging spot reports from multiple subordinates which overlap on the same enemy units. SA does not currently implement a sophisticated sensor data fusion algorithm, but the CE architecture supports the implementation of one.

In order to allow other CE component objects to retrieve SA state data, SA provides accessor functions to all of its information. To improve efficiency and reduce the computational load, the calculation of derived information in SA is mostly demand-driven; that is, it is only computed when asked for.

Examples of the types of information contained in SA include: the mission objective(s); location, strength, and type of known and suspected enemy units; location, strength, and composition of subordinate and peer units; location and type of all battalion and company control measures; location of known obstacles; and time remaining to complete the current mission.

4.5 Terrain and Environment

Terrain analysis is one of the most complex and critical components involved in C2 decision-making. In order for the CE to perform effective mission planning, execution, and tracking, it must continually evaluate the terrain and environment in which it is operating. The Terrain classes provide services to aid this evaluation and build the representation of the commander's perception of the environment.

The services provided by the Terrain classes are built on top of the CFOR Infrastructure's Environmental Utilities (EU) library. This library contains a set of basic terrain related services, and is based on a

tessellation of the underlying polygonal terrain. The EU utilizes this terrain tessellation to provide services which perform analysis of trafficability, fields of fire, cover and concealment, and line of sight. In addition, the EU provides services which allow access to the basic terrain data such as elevation, soil type, feature type, and coordinates.

The Terrain classes utilize the EU library to provide services to other CE components for analyzing the terrain at the individual vehicle and aggregate unit level. These services are used primarily by the constraint sets (described below) in the process of constructing a plan.

The Terrain services utilize information from SA to support their analysis. For example, known and suspected enemy locations are needed to generate covered and concealed routes, and a unit's composition is needed to compute the size of a battle position. In addition, the Terrain services utilize information from the CFOR Infrastructure's C2 utilities to support their analysis. For example, the range of a unit's weapons system is needed to generate attack by fire positions.

The services provided by the Terrain classes adhere to all control measures which have been specified in the order by the commander's higher echelon. This includes such control measures as unit boundaries, axes of advance, routes, assembly areas, and battle positions. In addition, known obstacles and other no-go areas are considered by these services. For example, if a unit is to defend in sector, then the sector boundaries must be honored in the generation of defensive battle positions and routes to subsequent battle positions.

Examples of terrain analysis services provided by the Terrain classes are the computation of: mobility corridors, based upon a unit's composition; avenues of approach, based upon an objective, a unit's boundary lines, and a unit's composition; routes, based upon enemy locations, a unit's composition, and time available to traverse the route; overwatch positions, based upon an objective location, enemy locations, weapons ranges, and a unit's composition; defensive battle positions, based upon an objective location, enemy locations, weapons ranges, and a unit's composition; assault positions, based upon an objective location, enemy locations, and a unit's composition; and attack positions, based upon a line of departure, objective location, enemy locations, and a unit's composition.

4.6 Planner

The Planner is the CE component which generates, evaluates, and selects a course of action that satisfies the mission objectives within the guidance specified by the higher echelon commander. It is invoked when a new order is received, or whenever the situation warrants a change in the course of action. The CE software uses an approach based upon constraint satisfaction to perform its planning. The following sections describe the critical components which are involved in the planning process.

4.6.1 Constraint Sets

The objects in the CE software which encode all of the knowledge required to plan a given task are called Constraint Sets (CS). Each CS is a C++ class which specifies how to make a coherent set of decisions. For a given type of operation, a CS specifies the relevant decisions to be made and generates options for each. The decisions can be sequential or parallel, and in any order. In contrast, finite state machines specify how to perform a temporal sequence of actions. The CE software utilizes two types of CSs: component CSs and composite CSs.

A component CS is used to evaluate alternatives and generate a feasible solution to an individual component of an overall mission. Each component CS contains specific knowledge to generate a plan for a particular task. There are many component CSs defined in the CE software, with each one typically corresponding to a single ARTEP task. For instance, there is a component CS which plans unit-level tactical movements, and another component CS which plans obstacle breaches. An instantiated component CS is a plan for executing a specific behavior given the current or expected tactical situation.

A composite CS is a collection of component CSs which are dynamically linked at run-time to form a mission. There is only one composite CS defined in the CE software, and it has no task-specific knowledge associated with it. Each mission will construct and generate a unique composite CS at run-time, which is capable of planning the mission at hand. The component CSs in the composite CS are linked together spatially by each of their start and end points. The composite CS also handles the allocation of mission-critical resources, such as time

and forces, across the component CSs. An instantiated composite CS is a plan for executing an entire mission over the course of time and space.

Both component and composite CSs have a common set of characteristics, as they are derived from a base class CS. All CSs have four basic components: a set of minimum input variables, a set of derived variables, generator functions, and prioritizer functions.

The set of minimum input variables are those pieces of information which are required to construct a plan for the given CS, and are supplied to the CS when it is initially constructed. These typically include the unit's name, the unit's composition, the known and suspected enemy locations, the start and end points, and the current order from the higher echelon commander.

The derived variables are those pieces of information which are computed by the constraint set. Each derived variable corresponds to a tactical choice which must be made by the CE. Once a set of consistent values have been generated for all derived variables in a CS, the CS represents a feasible plan for accomplishing the task at hand and is said to be instantiated. Examples of derived variables are the allocation of subordinate forces, route selections, tactical position selections, formation selections, and speed selections. The derived variables are ordered such that a given derived variable depends only on the previous derived variables.

Each derived variable has a corresponding generator function. A generator function produces a list of candidate values for a given derived variable. The values generated are consistent with the choices for previous derived variables and with the current battle state. Each value represents a different option for satisfying that derived variable. A variable can be of a simple type (such as a floating point number representing a speed) or a complex type (such as another constraint set). The generators invoke terrain analysis and situational awareness services as needed to support relevant decisions.

Each derived variable may also have a corresponding prioritizer function. A prioritizer function orders the values generated for a given derived variable in a best-first or least-constraining order.

As mentioned above, in some CSs a derived variable may be another CS, which returns different feasible

solutions to its parent CS. In this case, the CS is itself a generator function since it is dynamically generating multiple instances of itself as candidate values for the current variable.

The process of instantiating a constraint set involves exercising the generator and prioritizer functions until a consistent set of values for all derived variables has been found (e.g. a feasible set of firing positions, routes, etc. for an assault; a set of formations, sub-routes, and overwatch positions for a move). To develop a CS, a software engineer must specify in code the relevant variables, in what order they are to be matched, on which previous variables they are to depend, and how to generate candidate values for a variable given the current values of previous variables (if any).

4.6.2 Order to Constraint Set Decomposition

The CE software must examine the current order to determine what component CSs are needed to fully accomplish a given mission in accordance with the higher echelon commander's guidance. This function is performed in the CE by a class which decomposes CCSIL orders into an appropriate composite CS.

This decomposition software is based on concepts developed by Logicon RDA which break down Army missions into categories of tasks which can be performed to accomplish each mission type [Kleiner, et al, 1995]. Each mission which the CE can perform has a set of applicable ARTEP tasks which fall into one of the following categories: Achieve Tactical Disposition, Reduce Enemy Posture, Achieve Culminating Task, Consolidate, and Perform Situational Interrupt.

The CE software utilizes these concepts to decide which tasks, both explicit in the order and implied by the order, are to be performed to successfully execute the mission. It then maps these tasks into their corresponding component CSs and constructs a composite CS which contains these component CSs.

4.6.3 Constraint Satisfaction Tool Planner

4.6.3.1.1 Combinatorial Constraint Satisfaction

The Combinatorial Constraint Satisfaction (CCS) class is the CE component which invokes the generator functions of the CSs during the

instantiation process. It acts as an interpreter for CSs, handling the interactions between choices by searching the implicit space of possible choices. It successively calls the generator functions for each variable of each CS in the mission. If no values can be generated for a given variable which are consistent with the values selected for previous variables, then CCS backtracks to reconsider other values of the prior variables. Once CCS successfully instantiates a composite CS, the planning is complete.

The ability to embed CSs within one another, as mentioned above, is facilitated by the fact that CCS can recursively invoke itself. This, combined with the capability to dynamically link CSs together at run-time into a variable length composite CS, avoids a combinatorial explosion of the number of CSs to be developed. If this were not the case, pre-defined CSs for all mission possibilities would be required. Additionally, encoding the CSs as C++ classes allows for inheritance among common behaviors at multiple echelons, which lessens the number of required CSs and the amount of code duplication.

4.6.4 Replanner

When replanning is needed, CCS is invoked to plan reactions. When the reaction is complete, CCS is invoked to replan the remainder of the mission from the correct re-entry point. Reactive planning is complete on the order of seconds because the search space required for terrain analysis is small.

4.7 Plan

A plan in the CE is an object which consists of three component objects: the set of instantiated constraint sets, an execution matrix, and an operations order.

The plan will be followed by subordinates, and progress will be measured against it by the Tracker. It contains information which indicates the constraints which were used to generate particular nodes of the plan, for use in replanning and new OPOD evaluation.

4.7.1 Instantiated Constraint Sets

Once the planning process has been successfully completed as described above, a fully instantiated composite CS with fully instantiated CSs results. However, this representation of the plan as a set of

variables with corresponding values is not sufficient to describe the entire mission which has been planned for the commander's unit. It does not describe all details of the tasks which need to be assigned to subordinate units, only those characteristics which needed complex planning due to dependencies upon other values. Additional representations with more detailed information are needed and are described below.

However, these instantiated CSs contain the context in which the choices for the derived variable were made, and are therefore useful to save for future reference. In particular, their primary use is to assist the CE in performing partial replanning of the mission as the situation warrants.

4.7.2 Execution Matrix

After the CSs are fully instantiated, each CS generates its part of an object in the CE called the Execution Matrix. The Execution Matrix is a time-phased list of the company- and platoon-level tasks which the unit will perform as it carries out the plan.

Information contained in the Execution Matrix is the equivalent of that contained in an Army operations order execution matrix, plus detailed segmenting information needed for mission tracking and additional information required for execution of the plan by the CE. The Execution Matrix is designed to represent the flow-down definition of the mission from the form it takes in the instantiated CSs to a detailed, quantitative sequence that can be easily monitored and executed. As all tactical decisions were previously made by the CSs, the primary function of the Execution Matrix is to organize these decisions in such a way that they can be easily communicated to subordinate units and ensure synchronization.

The Execution Matrix representation is a hierarchy, the root of which is the mission itself. Lower levels of the hierarchy decompose the mission successively into phases, tasks, and segments. A data structure is also assembled in the Execution Matrix for the CE to use in monitoring events and issuing commands to subordinate units.

As each CS builds its part of the execution matrix, it also creates the segments that are part of each phase of the matrix. This segmentation scheme was inspired by the Autonomous Control Logic system [Glasson, 1992]. Segments are defined as portions of

a mission phase which have homogeneous attributes. Phases can be segmented in any way desired, but are typically segmented spatially or temporally. Each segment contains one or more transition arcs to other segments and may also contain expectations which embody the attributes of the segment. Some example segment attributes are: nominal times for beginning and end of segment; exposure state; likelihood of enemy contact; and unit formation.

4.7.3 Operations Order

After the Execution Matrix has been generated, it is turned into a CCSIL C++ class operations order or fragmentary order. This order is then converted into CCSIL for transmission to the subordinate units. This is the third and final component of the Plan. It is useful to save in order to perform simple replanning where the CE needs to make minor modifications to the previous order, such as changing a speed or formation. Having access to the order for reference allows for easy composition of fragmentary orders.

4.8 Mission Tracker

The Mission Tracker monitors the progress of the commander's unit as it executes its planned mission. It continuously compares actual states with expected states and initiates requests for replanning when corrective action is needed. It is also responsible for responding to or forwarding incoming messages from the CE's superior; relatively simple orders, such as Execute Directives, are handled by the Tracker directly, while more complex ones, such as Operations Orders, are forwarded to the Planner.

The Mission Tracker makes use of segments, described previously, to measure the unit's progress against the Plan. The Mission Tracker maintains the current segment for each subordinate, the unit as a whole, and the unit commander. Transitions between segments are detected using events, described above. When a segment transition event occurs, the Mission Tracker first checks that all execution state expectations (as described above, in the discussion on events) have been met. If they have, the Mission Tracker then queries the Plan for the next segment, transition events, and expectations for the affected unit(s). The new transition events are then registered with the Event Processor. Note that some segment transitions may require that the CE take specific actions (e.g., sending an Execute Directive to a

subordinate). This is handled by attaching an appropriate callback to the transition event. If all execution state expectations have not been met, the Mission Tracker invokes the Replanner to adjust the plan accordingly.

5. Future Work

Army CFOR CE development is currently in progress, and a variety of tasks are scheduled for the near future. These include continued expansion of the Army Armor Company Team CE, development of additional Army CE's (with horizontal expansion at the company level and vertical expansion to the battalion level), and refining and improving the basic CE architecture. In addition, the application of this CE software and architecture to a non-Army command entity is a mid-term goal.

5.1 Expansion of Army Armor Company Team CE Capabilities

The following additional Army Armor Company Team CE capabilities are planned to be developed in support of the STOW 97 exercise:

- Expansion and improvement of capabilities in attack and defend missions.
- Integration with an Army Fire Support Team (FIST) CE to plan and execute indirect fires in the offense and defense.
- Development of capabilities for planning and executing movement to contact missions.
- Integration with an Army Company Trains CE to plan and execute combat service support operations.
- Integration with an Army Rotary Wing Aircraft Company CE to plan and execute combined arms coordination.
- Development of capabilities for planning and executing engineer operations.

5.2 Additional Army Command Entities

The following additional CE's are planned to be developed in support of the STOW 97 exercise:

- Development of an Army Fire Support Team (FIST) CE.

- Development of an Army Battalion Cdr/S2/S3 CE.

- Development of an Army Battalion Fire Support Element (FSE) CE.

- Development of an Army Company Trains CE.

6. Conclusions

The architecture presented herein currently serves as the basis for a successful implementation of the C2 decision-making of an Army Armor Company Team Commander CE. This CE is capable of performing mission planning, execution, monitoring, and replanning. It has been successfully demonstrated at various STOW program events, and continues to be expanded and improved as software development proceeds. As the CE's capabilities are increased, the quality of its tactical decisions are also being improved. This CE architecture is rich and flexible enough to be applied to CE's at multiple echelons, as well as CE's in other service areas.

7. Acknowledgments

This work is sponsored under contract N66001-95-C-6006 from the Defense Advanced Research Projects Agency (DARPA). The authors wish to thank the members of the CFOR team at DARPA, NRaD, MITRE, and Logicon RDA for their dedication, support, and guidance.

8. References

Salisbury, Marnie R., Booker, L. B., Seidel, D. W., Dahmann, J. S., "Implementation of Command Forces (CFOR) Simulation," *Proceedings of the Fifth Conference on Computer Generated Forces and Behavioral Representation*, Orlando, FL, May 9-11, 1995, pp. 423-430.

Kleiner, Martin, Carey, S., "Company Team Command Forces, An Introduction to Decision Making," Version 1.0, Jan 5, 1995.

Glasson, Douglas P., "An Autonomous Control Logic Concept for the Autonomous Undersea Vehicle," American Institute of Aeronautics and Astronautics, 1992.

9. Authors' Biographies

Rob Calder is a Senior Software Engineer in the Technology Research Group at SAIC in Burlington, MA. He has been involved in the development of DIS CGF systems for over five years, and is the principal investigator on the CFOR project. Prior to joining SAIC, he was a software developer on multiple generations of CGF systems and the ModSAF project lead, at Bolt Beranek and Newman/Loral Advanced Distributed Simulation. His primary research interests are in the area of tactics and behavior representation and generation for computer generated forces. Mr. Calder holds a Master of Science degree in Computer Science from Boston University.

Rich Carreiro is a Software Engineer in the Technology Research Group at SAIC in Burlington, MA. He has been performing software design and development on the CFOR project for the past year and a half. Mr. Carreiro holds Bachelor of Science degrees in Electrical Engineering and Physics, both from the Massachusetts Institute of Technology.

Jim Panagos is a Consultant to SAIC in Burlington, MA. He has been involved in the development of DIS CGF systems for over 10 years, and is currently performing design and development on the CFOR project. His primary research interests are in the areas of tactics, behavior representation, and automated planning and generation for computer generated forces. Mr. Panagos holds a Master of Science degree in Computer Science from the Massachusetts Institute of Technology.

Rob Vrablik is a Software Engineer in the Technology Research Group at SAIC in Burlington, MA. He has been involved in the development of DIS CGF systems for over six years, and is currently performing design and development on the CFOR project. Mr. Vrablik holds a Bachelors degree from Dartmouth College.

Ben Wise is a Senior Scientist at SAIC's Technology Research Group in Burlington, MA. He received a B.S. in Physics from MIT and a Ph.D. in Engineering and Public Policy from CMU. He taught graduate operations research, probability and statistics, and artificial intelligence at Dartmouth College before entering the commercial sector. He worked on corps-level battle simulation and strike planning tools while at McDonnell Douglas, before

moving to BBN and assisting the SIMNET, Odin, ModSAF, and Warbreaker projects. He moved to SAIC in 1993 and started a new office specializing in advanced simulation and planning technologies. He served SAIC as the initial lead in the Corps Level Computer Generated Forces, Command Forces, and SAT/IAT projects. He is currently involved in several projects focusing on issues in the linkage of constructive and virtual simulations. His research interests focus on planning under competition and uncertainty.

Forrest Chamberlain is a Member of the Technical Staff in the Computer Generated Forces section at TASC. Forrest has been involved in CGF work since joining TASC in 1994. He is currently responsible for the terrain reasoning and mission tracking components of the CFOR project and is a critical contributor to the ICTDB terrain representation effort. Prior to joining TASC, he participated in the hardware and software design of a "wearable" computer system at Carnegie Mellon University, where he earned his Masters degree in Electrical and Computer Engineering. Mr. Chamberlain also holds a Bachelor of Science degree in Electrical Engineering from Cornell University.

Doug Glasson is a Department Research Analyst at TASC, Reading, MA, and is TASC's principal investigator on the CFOR project. His previous work in autonomous systems included Lead Architect for TASC's AUV Autonomous Control Logic Concept, and Program Manager of the Adaptive Tactical Navigator development. Mr. Glasson holds a Bachelor of Science degree in Aeronautical Engineering degree from RPI and an Engineer in Aeronautics and Astronautics from MIT.

Knowledge Acquisition and Delivery: Constructing Intelligent Software Command Entities

Seth R. Goldman
Hughes Research Laboratories
3011 Malibu Canyon Road, Malibu, CA 90265
seth@isl.hrl.hac.com

1. Abstract

In this paper, we discuss our general approach to knowledge acquisition and delivery and how we have applied it to the construction of intelligent software command entities as embodied in our work on the CFOR (Command Forces) and MC IC (Marine Corps Individual Combatant) DARPA programs. There are five key issues to address when integrating knowledge products with traditional software models: 1) Modularity - we don't want to have to get all the knowledge before the rest of the code can be developed, 2) Validation - the expert must be able to verify the acquired knowledge, 3) Scoping - we must be able to specify default parameters as place holders until the requisite knowledge can be acquired, 4) Reusability - decisions should be captured at the appropriate level of abstraction within and across domains, 5) Deliverability - the knowledge must be accessible to the software clients but independent. Each of these issues will be discussed in detail, together with examples of the knowledge bases derived for these DARPA CFOR and MC IC programs.

2. Introduction

For the past 18 months, we have been working on the DARPA CFOR (Command Forces) and MC IC (Marine Corps Individual Combatant) programs. In 1995, the CFOR team was tasked to develop a software command entity to model an Army Tank Company Team Commander. The MC IC team was tasked to develop a smart Rifle Squad leader within the ModSAF simulation environment. In 1996, the MC IC work continues and the CFOR team is tasked to develop a Marine Rifle Platoon Commander and a Marine Rifle Company Commander. There are many differences between the two programs but they share a common need for intelligent decision making to guide planning and behaviors. We have a great deal of experience in building complex knowledge based systems for a variety of applications (e.g., traveling wave tube design, financial analysis and investment). The current programs offered new challenges: to acquire the knowledge from a variety of sources (e.g., interviews with a subject matter expert (SME),

military training documents, documents compiled by the SME), and to make this knowledge available to a decision making process embodied in either the ModSAF simulation environment or our command entity software.

To meet these challenges, we have developed a mechanism called *fuzzy tables*, based upon our Modular Knowledge Acquisition Toolkit (M-KAT) methodology. The M-KAT methodology differs from traditional expert system construction techniques by emphasizing a very tight interview - implementation - feedback cycle. The knowledge acquisition process and supporting software environment facilitate rapid prototyping of the expertise so that the SME can quickly explore the knowledge within the overall domain framework. Fuzzy tables are an abstraction of some of the most commonly used parts of M-KAT. We expect that with training in the construction of fuzzy tables, domain experts will be able to *knowledge engineer* themselves and produce knowledge bases that can be integrated in a variety of applications.

Fuzzy tables are used in a variety of ways: 1) they direct the knowledge acquisition process and keep it focused, 2) they provide a declarative representation of the SME's decision making process, and 3) they serve as input to the fuzzy table runtime engine which provides client applications with access to the SME's knowledge via a query/response interface. Fuzzy tables are modular, verifiable, expandable, reusable and deliverable. Each of these properties is addressed below using examples from our CFOR and MC IC work. All the examples are drawn from the context of the appropriate military units conducting an attack. At the conceptual level, the doctrine for attack is not significantly different for an army tank company versus a marine rifle squad. Both units are concerned with finding good positions to launch the attack, finding good support positions for suppressing the objective, and responding to unexpected enemy encounters or to obstacles such as minefields.

3. Modularity

Divide and conquer is a well known technique in problem solving. Decomposing a problem into small, manageable pieces and combining the results produces a more robust solution which is easier to validate and maintain. In addition, the smaller pieces are potentially useful in solving other problems. We have applied the same concepts in our development of fuzzy tables. Each table documents a single decision made by the SME. Table 1 depicts the decision concerning the time constraints imposed by linking up with the main force at a particular rally point.

time-to-new-rp	time-remaining	new-rp-time-constraint
10min	<30min	slightly-constrained
10min	2hr	not-constrained
10min	>5hr	not-constrained
30min	<30min	not-possible
30min	2hr	slightly-constrained
30min	>5hr	not-constrained
2hr	<30min	not-possible
2hr	2hr	not-possible
2hr	>5hr	slightly-constrained

Table 1: Rally Point Time Constraint

Fuzzy tables consist of a series of input columns followed by a single output column. Each column represents a factor that the SME considers in making the decision represented in the final column. In this example, the decision is called *new-rp-time-constraint* and has one of the following values: *slightly-constrained*, *not-constrained*, or *not-possible*. The decision is based upon two factors: 1) how long will it take to get to the new rally position and 2) how much time is left in the mission. Based upon these two factors, a decision is reached which will then be used to make other decisions as we shall see below.

A key feature of fuzzy tables is that both the inputs and outputs need not be absolute values but can instead be fuzzy values. For example, the time to the new rally position might be one hour. In this case, the one hour will get translated into a fuzzy value of 70% 30min and 30% 2hr. To paraphrase, one hour is mostly like thirty minutes and a little bit like two hours. If the time remaining is two hours then the result from the table will be 70% slightly-constrained and 30% not-possible.

Where does the knowledge come from, how do we map the absolute values to fuzzy values? The knowledge is elicited from the SME through one or more interviews. In our original implementation of fuzzy tables, the mapping from absolute values to fuzzy values was represented by a separate series of rules. Maintaining a separate knowledge source was cumbersome and led to inconsistencies over time. Therefore, we augmented the fuzzy table representation to include not only the knowledge of how to map absolute values to fuzzy values but also the enumeration of legal fuzzy values for each column of the table. We still use rules to perform the actual mapping but these are generated automatically when the fuzzy tables are parsed. In this way, all the knowledge required for a particular decision is represented by a single fuzzy table. We shall see examples of this below.

How do we combine fuzzy tables to make more complex decisions? The output column for a fuzzy table can be linked to an input column of other fuzzy tables. Thus, decisions can be used as inputs for other decisions. For example, the output from Table 1 is used as input in deciding the method of attack for the rifle squad.

The decision of how a rifle squad should attack an enemy has the following possible outcomes: na - cannot carry out the attack, ab - abort the mission (the costs are too high), mv - move to a new location and reconsider, fr - conduct a frontal assault, se - conduct a single envelopment by establishing a suppressive base of fire (BOF) position. Entries in the output column separated by slashes indicate alternatives that cannot be distinguished by this table. Additional knowledge is required to choose one outcome over the other.

fire-teams-left	bof-and-assault-position	bof-ability	new-rp-time-constraint	method-of-attack
1	yes	yes	not-possible	na
1	yes	no	slightly-constrained	fr/ab
1	no	no	constrained	ab
1	no	no	slightly-constrained	ab/mv
2-or-3	yes	yes	constrained	se
2-or-3	yes	no	not-constrained	fr/ab
2-or-3	no	yes	not-possible	fr/ab
2-or-3	no	yes	constrained	mv/fr
2-or-3	no	yes	not-constrained	mv

Table 2: Rifle Squad Method of Attack

We see in Table 2 that one of the input columns is the output from Table 1: Rally Point Time Constraint. Decisions can thus be decomposed into a sequence of easier decisions. This modularity facilitates the acquisition, testing, and maintenance of complex decisions.

4. Validation

So where does the input come from when it is not from other tables? The answer depends upon how the tables are being used. Because the knowledge engineer must write code to implement the answers to questions relevant to the problem, this code generally requires information from the client application. When the tables are being used by the client application, the code for a particular column is executed, the result returned by the client is then used as the input for the column. Returning to Table 1, the first column is the time required to get to the new rally point (RP) from base of fire position (BOF).

This datum must come from the client application. The client application must supply a *callback* routine which computes this value. The knowledge engineer then writes a small piece of code (*glue*) to call this routine with the appropriate parameters, in this case, the parameters are the BOF and the RP.

If the tables are being used to debug the knowledge acquisition process, then instead of calling back to the client application, we want to ask the expert (SME) to provide the required data. Our fuzzy table implementation provides tools that support both modes of operation. The knowledge engineer can write *glue* routines that will either callback to the client application or ask the expert depending upon the context.

One of the drawbacks of our initial implementation was that many of the details were hidden from the expert; the tables did not contain sufficient information by themselves. The knowledge engineer had to do some programming to make the tables operational. Since some of the knowledge was embedded in the code, validation was more difficult. We have addressed this issue by expanding the column headers for the table so that they include all the information necessary to operationalize the tables without programmer intervention. This does not free the knowledge engineer from writing the glue routines; however, it does make explicit the parameters to those routines and the legal values they may return.

Table 3 shows the decision of what formation the tank company should use. An asterisk in a cell indicates that the answer doesn't because other factors control the outcome. To provide better information, the column headers have gotten a bit more complicated. They now contain information about permissible values and how they are to be computed. Each header consists of three elements: a name for the decision, the value specification, and the

(current-action (movement assault flank-security) (lfor:current-action! ->unit))	(enemy-contact (likely possible unlikely) (lfor:enemy-contact! ->unit))	(company-formation (line echelon column wedge vee) (:movement-module :company-formation ->unit))
assault	*	line
flank-security	*	echelon
movement	likely	vee
movement	possible	wedge
movement	unlikely	column

Table 3: Company Formation

path. The last column in Table 3, which provides the outcome of the decision is:

name:	company-formation
value spec:	(line echelon column wedge vee)
path:	(:movement-module :company-formation ->unit)

The value spec for company-formation limits the output to be: line, echelon, column, wedge, or vee. No other values are permitted. The path specifies how this table is invoked and the parameters it requires. Parameters are indicated by names beginning with “->”. The company formation table requires only a single parameter, the unit name of the company. For the inputs of Table 3, the first column is:

name:	current-action
value spec:	(movement assault flank-security)
path:	(lcfor:current-action! ->unit)

When the first element of the path is enclosed in vertical bars, that indicates a callback to the client application. In this case, the fuzzy table determines the current action of the unit by asking the client application to compute the value and provides the name of the unit as a parameter. During validation, the user would be prompted to select one of the possible values.

The second column is also a callback, asking the client application to determine the likelihood of enemy contact for the unit:

name:	enemy-contact
value spec:	(likely possible unlikely)
path:	(lcfor:enemy-contact! ->unit)

In most circumstances, this question probably requires additional knowledge and more reasoning. However, it can be initially implemented as a

callback to facilitate development of the client application. When additional knowledge acquisition yields more details concerning this decision, new tables can be built and used in place of this callback. This ability to incrementally expand the scope of the reasoning capabilities of the knowledge base is crucial, and is one of the features of M-KAT.

5. Scoping

In our experience, constructing a command entity requires concurrent development of the knowledge base and the client application. It is essential to minimize the interdependence of these development paths. We use a top-down knowledge acquisition process to achieve this goal. When interviewing the SME, we attempt to identify the major high-level decision points that guide the planning process. With these decisions in place, the client application can continue testing and development while we work with the SME to elicit the lower level decisions which feed the high-level decisions.

This approach serves us well for two reasons: 1) as stated above, the impact upon development of the client application is minimized, and 2) access to the SME is often restricted to discrete intervals. As the SME becomes familiar with the fuzzy tables and our methodology, we can perform knowledge acquisition interviews over the phone in a short amount of time. Fuzzy tables can be generated by the interview process and then sent to the SME to be filled out.

Scoping also helps keep the SME focused on the particular decision at hand instead of becoming distracted by the details of the input parameters. For example, if one of the columns deals with how far away the unit is from the objective, we can simply characterize the distance as: near, medium, or far. Later on, we will ask the SME to specify how those fuzzy values relate to actual distances.

(c+c (no 0.0 0.4) (yes 0.6 1.0)) (lcfor:c+c-positionl- >ap-loc ->ep-loc))	(distance ((very-close 0 200) (close 250 1200) (far 1500 :+infinity)) (lmodsaf:distancel - >ap-loc ->ep-loc))	(add-sbf-sites? (yes no) (lcfor:add-sbf- sites?!->ap-loc - >ep-loc))	(ap-to-obj- exposure ((small 0 40) (medium 45 55) (large 60 100)) (lcfor:exposurel - >ap-loc ->ep-loc))	(viability (very-good good fair poor unacceptable) (:ap-evaluation- module :dismounted- viability ->ap-loc - >ep-loc))
yes	very-close	yes	small	very-good
yes	close	yes	medium	good
yes	far	no	large	poor
no	very-close	yes	small	poor
no	close	yes	medium	poor
no	far	no	large	unacceptable

Table 4: Dismounted Assault Position Viability

The mapping from absolute to fuzzy values is expressed in the value spec part of the header.

```

name: distance
value spec: ((very-close 0 200)
             (close 250 1200)
             (far 1500 :+infinity))
path: (lmodsaf:distancel ->ap-loc ->ep-loc)

```

For column 2, distance, very-close is anything between 0 and 200 meters, close is between 250 and 1200 meters, and far is anything more than 1500 meters. Notice that the values don't fully cover the range of numbers which raises the question of how is 225 meters going to be represented? The fuzzy table software automatically interpolates and assumes that the point halfway between two values will be half one fuzzy value and half the other. Thus, 225 meters is 50% very-close and 50% close.

6. Reusability

At the lower echelons, such as tank companies and rifle platoons, the doctrine for conducting an attack on an objective is very similar. We would like to be able to exploit this aspect and reuse some of the tables developed for one application in another. If the decisions and their inputs are indeed shareable, there is still one aspect that will almost certainly be different, the value specifications in the tables. These specifications determine the mapping from absolute values to fuzzy values. Using the example from Table 4, the values used to convert absolute distances to fuzzy distances would be different for a rifle squad compared to a tank company. Infantry traveling on foot will consider 1000 meters to be much farther than if they were mounted in a tank.

For now, we lack an elegant solution to this problem.

The simple solution is to copy the table and change the values appropriately. At least the work of validation and acquisition are capitalized upon. An alternate solution would be to leave it up to the client application to compute the fuzzy values. This places a large burden on the application developer and requires more knowledge to reside in the client application. A more satisfactory solution is to augment the value spec representation to include a reference to the client for each mapping. This would permit using the same tables for different clients and have the values mapped properly. This is probably the approach we will take when this issue gets addressed.

7. Deliverability

The current fuzzy table environment is implemented in Common Lisp and runs on Macintoshes, Suns, and SGIs. The Lisp environment provides easy interactive debugging of the knowledge as it is acquired. Clients connect to a knowledge server running in Lisp using TCP/IP sockets. While the Lisp environment is essential to the acquisition/development cycle, it is too limiting during execution of the client application.

We have looked at various ways to deal with the issue of providing the knowledge to the client application in an efficient manner. Ideally, the client developer should be able to link the knowledge base into the application directly or communicate with a knowledge server somewhere on the network. We are building a version of the fuzzy table runtime engine in C using a Common Lisp to C translator. In this approach, the knowledge engineer acquires and debugs the knowledge using the Lisp fuzzy table development environment, translates the Lisp code to C, and then compiles the C code into a library to be linked with the client application or run as a

standalone server process.

The knowledge engineer will deliver a compiled knowledge base along with the specification of all the queries (fuzzy tables) that can be handled together with the required callback that must be supplied by the client application. The application developer can access the knowledge base directly by linking it with the application or over the network by running it as a server.

This solution still requires that the knowledge engineer (fuzzy table developer) still have access to a Lisp environment. While it is possible to develop a complete user interface to the compiled knowledge base, we feel it is important to maintain the flexibility provided by the Lisp development environment. The knowledge engineer can quickly write code to modify the results of tables if necessary. For example, rather than build a new table to disambiguate the inconclusive outcomes from Table 2: Rifle Squad Method of Attack (e.g., mv/fr, mv/ab), it is often simpler to write a small piece of code to resolve the ambiguity. This applies during the knowledge acquisition and development process. In the final version, the disambiguation should in fact be done with a table.

8. Conclusions

Fuzzy tables provide a compact representation for knowledge captured from a domain expert. Their modularity makes it easy to break down the decision making process into manageable parts. Our ability to rapidly make the tables operational provides the SME with quick feedback and facilitates the validation process. In addition, the augmentation of the column headers provides explicit documentation of all the knowledge for a particular decision. Incremental expandability enables us to model the decision making process in a top-down manner, capturing the big picture decisions at first and later on focusing in on the details. This speeds up the development process and keeps the SME focused on the decision at hand. With some additional development work, we should be able to reuse tables easily where the decisions and their inputs are the same across applications and the variations are restricted to the mapping of absolute to fuzzy values. Finally, fuzzy tables can be delivered as C code which can be either compiled into an application or executed as a standalone knowledge server providing a high degree of portability and performance.

9. Acknowledgments

Portions of this work were supported by DARPA under contract DAAE07-92-C-R007, administered through TACOM, and N66001-95-C-6008 administered through NRaD. We would like to thank Charles Dolan, the co-developer of and driving force behind M-KAT, for his knowledge engineering assistance. We would also like to thank our SMEs, James Sinnott, Mack Brewer, and Scott Carey, without whom we would have no expertise to capture. The interest and support provided by CDR Peggy Feldmann, DARPA Synthetic Forces PM; Susie Hartzog and Jeff Clarkson, NRaD Project Managers, is greatly appreciated.

10. Biography

Seth R. Goldman is a Member of the Research Staff in the Information Sciences Laboratory at Hughes Research Laboratories. Mr. Goldman is a Ph.D. candidate in Artificial Intelligence at UCLA having earned his M.S. in Artificial Intelligence from UCLA in 1986 and his S.B. in Computer Science from MIT in 1982. He is the principal investigator on an NGIC contract to study methods of operationalizing OPFOR knowledge. He is the co-developer of M-KAT (Modular Knowledge Acquisition Toolkit). His interests include making application software smarter through knowledge integration, natural language understanding, and human-machine interfaces.

Task-decomposition Planning for Command Decision Making

Jonathan Gratch
University of Southern California
Information Sciences Institute
4676 Admiralty Way, Marina del Rey, CA 90292
gratch@isi.edu

1. Abstract

Developing synthetic command entities requires addressing a host of issues not normally faced by more traditional unit_level agent models. Commanders must reason over a broader scope and about events unfolding over long periods of time. Key amongst this is the ability to reason about and control unit interactions: coordinating the behavior of subordinates, meshing with the intentions of ones superiors, and managing the interactions with enemy forces who are intent on disrupting the commander's intentions. We describe the implementation of a rotor_winged aircraft (RWA) company command entity implemented in Soar and simulated within DIS. The command entity adopts planning techniques to manage the issues of coordination, control, and replanning that arise in this domain.

2. Introduction

As work in computer generated forces has developed, it has become more ambitious in its scope. A recent important effort is the development of so_called command forces or CFORs (Salisbury *et al.* 1995). The goal of the command forces project is to explicitly model command and control decisions in simulation. In contrast to the issues faced by vehicle or platoon level units, CFORs must model the decision making from a broader perspective and over longer time scales. Whereas vehicle_level decision making tends to be more reactive in nature, higher echelon units must deliberate about alternative courses of action, project effects into the future, and detect harmful (or beneficial) interactions between subordinate units and enemy forces.

In this paper, we describe the Soar/CFOR command forces project currently under implementation as part of the CFOR effort. Soar/CFOR extends the the Soar/IFOR capabilities to higher echelons and incorporates the communication and command

functions necessary to operate at these levels. The project is associated with the Synthetic Theater of War (STOW) program and is being developed in conjunction with the Soar/IFOR project (Laird et al., 1995, Rosenbloom, et al., 1995). Soar/IFOR is an implemented system for controlling intelligent pilot agents for participation in simulated battlefield exercises. Soar/IFOR has already participated in simulated combat exercises with expert human pilots, including the STOW_E and ED_1 exercise and will participated in the upcoming STOW97.

The initial implementation of Soar/CFOR has focused on the command functions of an AH-64 Apache attack helicopter company commander. Subsequent work will extend this functionality to the battalion level. Command behaviors include the ability to receive orders from one's superiors (live or simulated), plan missions for subordinate units, develop a situational awareness of the battlefield, monitor the execution of plans, and perform replanning whenever the situation dictates.

Soar/CFOR is developed within the Soar architecture which also serves as the system underlying Soar/IFOR agents. The demands of command decision making have led to considerable differences in the higher_level organization of CFOR agents when compared with Soar/IFOR agents. The greater focus on temporal and interaction reasoning has led us to draw substantially from the AI planning literature in the course of the command entity development. In particular, the Soar/IFOR entities, though they do have deliberation capabilities, are more focused on reaction than planning. Vehicle_level behavior is not guided by an explicit representation of the situation, but is rather implicit in rules that key off of the content of the current situation. This makes Soar/IFOR efficient and responsive to dynamic changes, but makes it more difficult to reason about interactions and changes

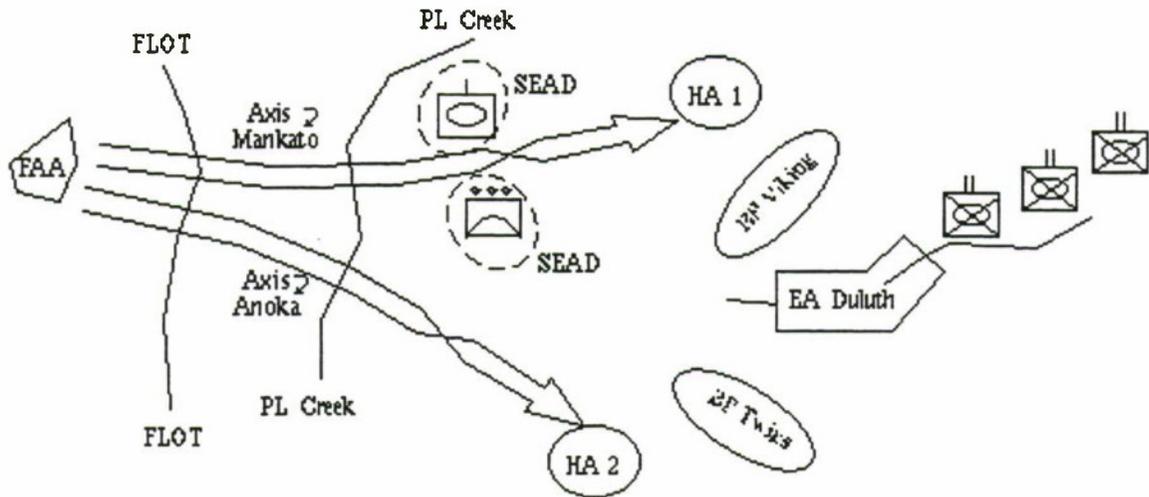


Figure 1: Attack Helicopter Bn attacks to destroy the 501 MRR in EA DULUTH

over time. In contrast, the command entity incorporates a reasoning style known in the planning community as hierarchical task-decomposition planning (Stefik 1981, Erol, et al., 1994, Ambros-Ingerson and Steel, 1988). Task-decomposition planners view a plan as a sequence of tasks, with dependency information that records the interactions and causal connections between tasks. Planning proceeds by taking individual tasks and decomposing them into a partially ordered sequence of more specific tasks, in response to the current situation. Task-decomposition planning meshes well with the hierarchical flavor of military decision making as well as the hierarchical structure of the Soar/IFOR agents which the Soar/CFOR entity commands.

3. Command and Control Requirements

The responsibilities of a command entity differ markedly from those of lower echelon units. This can be seen clearly by considering a typical mission flown by an Apache attack helicopter company. This example is based on the virtual Situational Training Exercise (vSTX 2) provide by Logicon, which served as the basis of a recent evaluation of our CFOR effort. The exercise has been generalized slightly to include capabilities we are expected to provide in STOW97. Figure 1 illustrates the operation overlay for a deep strike mission against enemy units.

In this mission, the helicopter company receives a mission from its battalion commander. In this case, the 1-155th Attack Helicopter Battalion is ordered to destroy the 501st Motorized Rifle Regiment as it passes through engagement area Duluth. This is a phased attack with company A moving along axis Mankato and company B moving along axis Anoka. This is a deep operation, meaning it is well beyond the forward line of our own troops (FLOT). Company A must pass through a known group of enemy forces and an artillery strike will be called to create suppression of enemy air defense (SEAD). Each company will proceed along their respective axes to a holding area. On orders from the battalion commander they will enter the battle position and commence the attack. Company A will attack first, and then coordinate a transfer of the engagement area over to Company B. Companies should report the crossing of all phase lines, bypass all enemy units, and report units of company size or greater.

The battalion commander transmits this mission to his company commanders. In our simulation, missions are communicated using the Command and Control Simulation Interface Language (CCSIL) developed by Mitre (Salisbury *et al.* 1995). CCSIL provides a structured language to facilitate all communication between CFOR entities. The mission includes information necessary for the company commander to perform mission planning: the goal of the mission (attack to destroy the 501 MRR); the actions to be performed by the battalion and brigade;

expected actions of enemy forces, plans for each company; and the operation overlay. Each company's plan is specified by a sequence of tasks. These are to be interpreted as high_level guidance or constraints on how the company commander develops his course of action. In this case, Company A is ordered to 1) move to and occupy battle position Viking along axis Mankato; 2) on order, destroy targets in engagement area Duluth, and 3) return to FAA and prepare for future operations. The mission also includes reporting requirements (e.g., report crossing of all phase lines) and coordinating instructions (e.g., coordinate SEAD).

3.1 Abstract and Implied Tasks

The tasks in the company orders are quite different in character from those typically given to simulated forces. The first difference is that the tasks are specified too abstractly to be directly executed. For example, moving to and occupying a battle position involves multiple tasks. Since the axis crosses the FLOT, the commander must coordinate a passage of lines with friendly ground forces. Different formations and speeds will be chosen for different points along the axis. The axis itself is an abstract construct and must be refined into a route based on characteristics of the terrain. Firing positions must be selected within the battle position, flanking positions selected, etc.

A second key difference is that the mission may contain many implied tasks. As a simple (but common) example, the axis may not go completely from the FAA to the holding area. The command entity must recognize whatever gaps exist and plan routes to fill in these missing pieces. More generally, the commander must deal with a whole host of issues involved in interpreting the mission statement and that are resolved by the principles of METT-T, but also involve considerable "common sense" reasoning. As another brief example, in one of the missions in which we participated, the axis of advance was specified in the reverse direction from what we were expected to fly. A human command would easily recognize that the direction should be reversed. Simulated command agents must be able to handle similar complications.

3.2 Managing Interactions

A large portion of the commander's planning focuses on managing interactions with other entities. With friendly units, the commander must insure proper

coordination: in the above mission, the commander of Company A must plan coordinate activities with ones' superiors, with other friendly units and between elements of his company. Coordinating with superiors requires reasoning about activities of higher echelon units. Other outside interactions include the coordination of passage of lines with ground forces, coordination of SEAD with division artillery, and the transfer of the engagement over to Company B. Each of these interactions places constraints on his mission planning, particularly timing constraints, which may influence how the plan is developed (e.g., what formations and speeds to use at different points of the mission).

Within his company, the commander coordinates interactions between subordinates. Scouts must be overwatched; rally points must be established if the company becomes separated. During the attack, several coordination issues arise. The commander must insure that units distribute their fires across the engagement area and adjust the company's position if units are interfering with each other or are coming under effective counterattack.

Perhaps the most complex interactions involve enemy forces. At the very least the commander must ensure his forces reach the battle position when the enemy is in the engagement area. Beyond this, the commander must recognize and manage potential threats the successful completion of his mission. This can be preplanned to some extent (e.g., planning secondary battle positions, rally points, etc.) but also may require replanning during mission execution (e.g., developing a route to bypass unanticipated enemy forces).

3.3 Replanning

As just alluded to, one of the most difficult requirements on command entity behavior is the need to handle unanticipated contingencies. The battlefield is a dynamic environment. Unanticipated contingencies can be handled with local reactions only to an extent. Often a plan has tight constraints and avoiding an unexpected enemy force early in the plan may have consequences for subsequent execution. Changes might be as minor as changing speed to as demanding as replanning the mission from scratch based on new information enroute. A command entity must recognized the interdependencies of plan steps in order to respond to such dynamic changes.

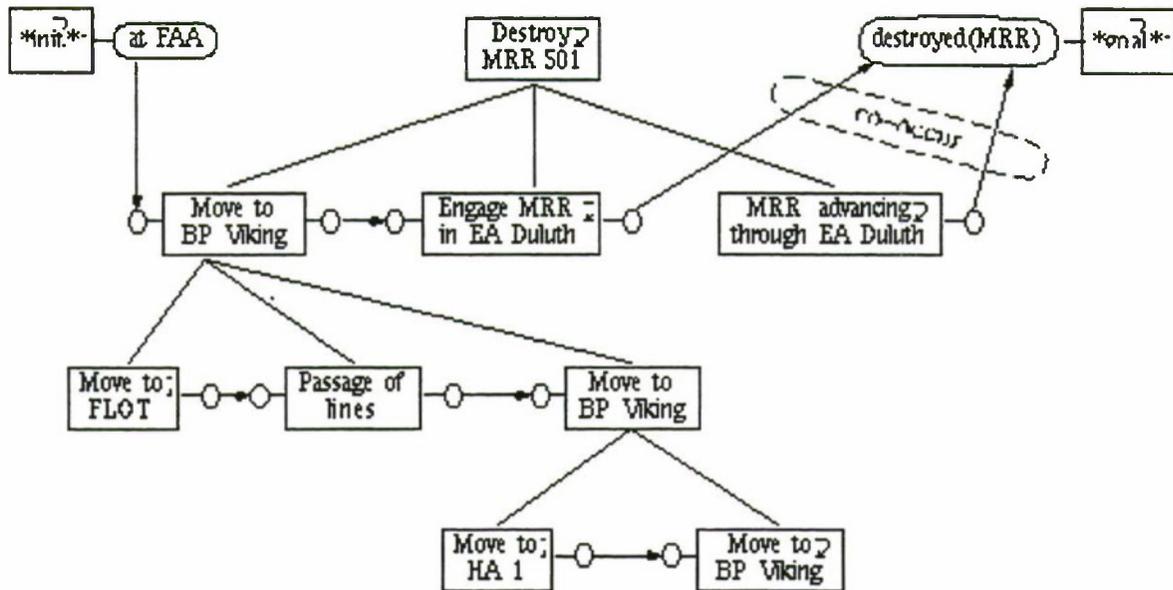


Figure 2: Hierarchical task network for (part of) the mission in Figure 1

4. Command and Control in Soar/CFOR

The above requirements place several constraints on the development of a command entity. The company commander must reason about interactions between his subordinates as well as with other forces. Besides constructing the initial plan, the commander must track the interdependencies between tasks, recognize how the changing situation effects these dependencies, and repair the plan whenever these dependencies are violated. To address these requirements, we adopted a plan representation known as *hierarchical task networks* (HTN) (Sacerdoti, 1977; Tate, 1977; Wilkins, 1988). Planning in Soar/CFOR is accomplished through a combination of techniques developed for HTNs and techniques developed in the *partial_order planning* paradigm (Chapman, 1987; McAllester and Rosenblit, 1991). First we will describe the plan representation.

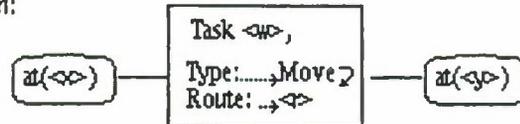
4.1 Plan Representation

Figure 2 illustrates a hierarchical task network for the ingress part of the mission. The egress part of the mission is left out for simplicity. The network represents a hierarchy of tasks. At the top of the hierarchy is the abstract task "destroy the 501st MRR." This is broken down into a partially_ordered

sequence of subtasks, two of which are to be performed by Company A and one of which corresponds to the actions of the enemy regiment. Subtasks may be further subdivided into more subtasks. The intended interpretation of the network is that subtasks represent a more detailed specification of how a task is accomplished. A task with subtasks is said to be an *abstract task*, and the subtasks are said to be a *decomposition* of the abstract task. Tasks that cannot be further decomposed are referred to as *primitive tasks*. These typically correspond to actions that can be directly executed by the agent. (Note that a primitive task at one echelon may be an abstract task at lower echelons. Primitive tasks for the company commander are converted into a set of task by the Soar/IFOR entities.) In Figure 2, tasks are represented as rectangles. Shaded rectangles correspond to primitive tasks.

Much like other plan representations, tasks in a hierarchical task network may have preconditions and effects. Preconditions are facts which must be true in the world to execute the task. Effects are those facts that are added or deleted by executing the action. In our plan representation, preconditions and effects may be predicates with an arbitrary number of variables. Currently, we do not implement variable quantifies in.

IF $\langle w \rangle$ is a task of form:



AND $\langle v \rangle$ intersects the FLOT at $\langle v \rangle$

THEN create subtasks:

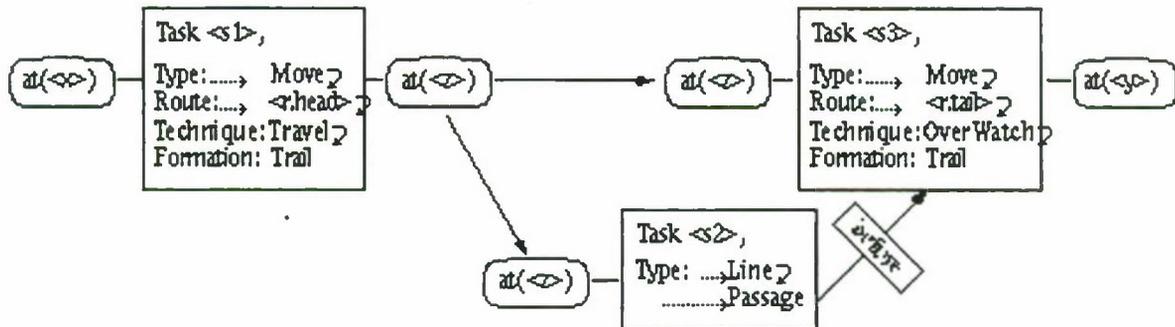


Figure 3: Task-decomposition operator

preconditions. In Figure 2, preconditions and effects are represented as ovals.

Typically the network will include two “dummy” tasks. The **init** task asserts all the facts of the initial state as its effects. The **goal** task has the overall purpose of the mission (destroy the 501st MRR) as its preconditions.

Dependency information is represented as links between preconditions and effects in the network, which are represented as arrows in the network. For example, for the enemy to be destroyed, the company must perform the engage task at the same time that the 501st is in the engagement area. This is indicated by a dependency of the destroyed precondition and the effects of these two tasks. Links between preconditions and effects are a special case of a more general concept of a *protection constraint*. If task1 asserts fact A which is a precondition to task2, the commander must ensure that fact A remains true from the end of task1 to the beginning of task2. This can be stated as a constraint that A must remain true from task1 to task2. Note that these protection constraints also force orderings between tasks: the asserting task must precede the task whose precondition it establishes. The planner may also impose ordering constraints directly between tasks. The hierarchical task network representation makes it

easy to express a variety of constraints on the plan structure.

4.2 Plan Generation

Planning is accomplished in a fashion similar to the IPDM architecture (Ambros-Ingerson and Steel, 1988). A final plan is developed through a process called *refinement search* (Kambhampati *et al.*, 1995). Initially one will start with a partial task network probably consisting of a few abstract tasks (in the mission described above, the commander receives the top node in the hierarchy and the first level of subtasks in the operations order). Typically, the initial plan cannot be executed. It may contain non-primitive tasks or tasks may have unsatisfied preconditions. The limitations in the initial plan are addressed by applying operations that modify the plan structure and, hopefully, result in a complete sequence of primitive tasks that achieve the goals of the mission. These planning operations are called *refinements* and can be classified by the type of limitations they address.

Task_decomposition: Non-primitive tasks are addressed by a refinement called *task_decomposition*. Task_decomposition operators specify how an abstract task might be broken down into a partial sequence of subtasks. Such an operator is illustrated

in Figure 3 (the syntax “<x>” denotes a variable named *x*). This rule checks if a movement task goes along a route that intersects the FLOT. If so, it creates three subtasks, the first moves along the route up to the FLOT, the second performs the passage of lines, and the third moves along the remainder of the route. This rule also augments the subtasks with movement techniques and formations appropriate to the crossing of a FLOT. Note that the decomposition involves more than simply asserting a set of subtasks; it may additionally assert ordering and protection constraints to the plan structure.

Establishment: Unsatisfied preconditions are addressed by two different refinements. The first is called *simple establishment*. This operation looks for some effect already in the plan structure that satisfies the precondition. If such an effect exists, a dependency link is created between this effect and the precondition. The effect is said to *establish* the precondition. This also enforces a protection constraint on the plan - no other task may delete this effect until after the task whose precondition is established by it.

If no existing effect in the plan can establish the precondition, an alternative method of establishment can be used called *step addition*. This operation adds some task to the plan that has an effect which unifies with the precondition. A link is then drawn between the effect of this new task and the unsatisfied precondition.

Protection Violation: A final class of refinements addresses potential violations to the protection constraints in the plan. For example, consider that effect *At(HoldingArea)* is protected from task1 to task2, that task3 deletes *At(HoldingArea)*, and that task3 can possibly occur between task1 and task2. In this case, there are two ways to refine the plan to remove this potential conflict. *Promotion* asserts an ordering constraint which forces task3 to occur after the protection interval (after task2). *Demotion* asserts an ordering constraint which forces task3 to occur before the protection interval (before task1). Finally, *separation* asserts a binding constraint which states that <x> cannot equal <y>.

Planning proceeds by incrementally applying refinements until a complete plan is discovered. Alternative refinements can be explored by depth_first search. If no refinements can be applied or there is an unresolvable flaw in the plan, the planner is forced to backtrack. Multiple courses of

action can be entertained by exploring different refinement sequences in parallel. Typically task_decomposition refinements are applied first, to sketch out the basic structure of the plan. Next, simple establishment, promotion, demotion, and separation are considered, Finally one considers step_addition.

4.3 Plan Execution

In addition to refinements, we implemented two other planning options. The command entity can initiate the execution of a task or terminate the execution of an executing task. Tasks may be executed if their preconditions are satisfied and no other unexecuted task precedes them. Tasks may be terminated if some prespecified termination criteria has been reached (e.g., a movement task terminates when the movement objective has been attained). The command entity may interleave execution and termination with the other refinements and thus achieve an interleaving of planning and execution.

4.4 Replanning

Replanning occurs in much the same way as the plan was initially developed. During the course of plan execution the current state may change in ways that violate or potentially violate the dependencies in the plan structure. For example, the execution of a task may not have the expected effects, or some unanticipated event may occur, such as a change in the location of the target. When such situations arise they are interpreted by the planner as limitations in the current plan, and are addressed by the same refinements used in plan generation. When preconditions become unsatisfied the planner will try to reestablish them through simple establishment or step addition. When that fails the planner will be forced to backtrack across the refinements that introduced the unsatisfied preconditions. When all else fails the commander can contact his superiors for further instructions.

5. Company Organization

In our simulation, we have made a distinction between the Soar/CFOR command entity, which does mission planning/replanning and the Soar/IFOR vehicle entity which implements the vehicle level behaviors. A human company commander must play both roles; he or she must plan the mission and control the vehicle, often both at the same time. We

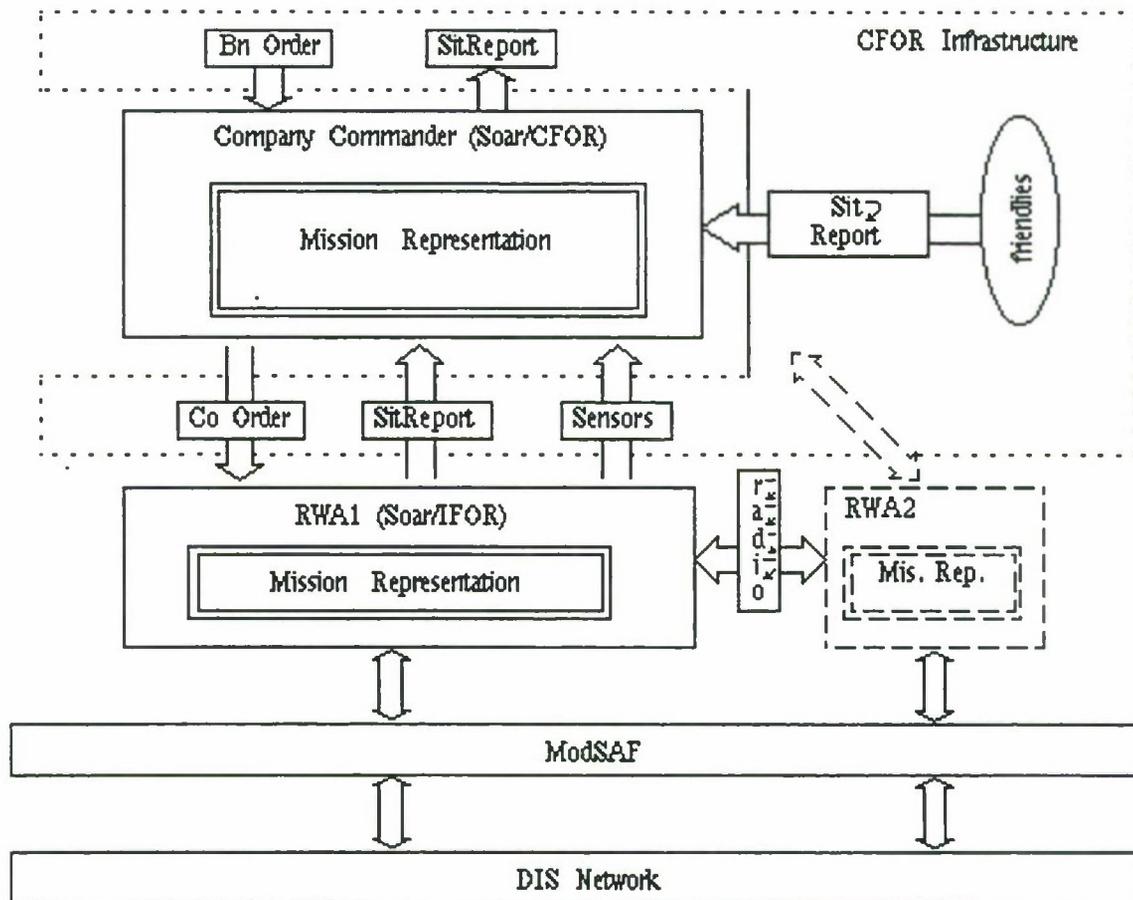


Figure 4: Company Organization

have addressed this problem by essentially dividing the commander's brain in two. Although they are separate processes, the Soar/CFOR entity is associated with a particular Soar/IFOR vehicle and has special information links to it.

Figure 4 illustrates the basic organization of a RWA company as simulated within DIS. The command entity is controlled by Soar/CFOR. Each RWA in the company is controlled by Soar/IFOR. The command entity is associated with a particular RWA and gains access to that vehicle's sensors through the CFOR infrastructure provided by Mitre. Communication between the command entity and the vehicles of the company (including his own) occurs via CCSIL messages. Vehicles may communicate directly to each other via CCSIL or simulated radio. The commander mediates all communication with units outside the company.

Initially the company commander receives a mission in the form of a CCSIL operations order.

Soar/CFOR develops a plan, backbriefs it to the battalion commander, and if approved, broadcasts the plan, via CCSIL, to the company. At this point all entities have a consistent representation of the mission. As the mission progresses, new information may become available: new information may arrive in the form of new orders, vehicle sensors, or situation reports (in CCSIL) from other units. It is the responsibility of individual vehicles to inform the command entity of relevant new information. If this information invalidates the current plan, the command entity will regenerate a new course of action and broadcast the new mission to the company.

An issue raised by this organization is how to model transfers of command, as when the commander is killed during the mission. The Soar/IFOR entities model a chain of command: when it becomes known that the commander is dead, the next vehicle in the chain assumes the commander's role. What this means in our company organization is that the new

commander acquires the interface to the command entity. Currently, we are not planning on modeling the loss of information and expertise that accompanies such a change.

6. Project Status

As of the writing of this article the Soar/CFOR command entity has been in development for eight months. The basic plan generation capabilities are in place and performed successfully during a recent evaluation based on a reduced version of the scenario described in Section 3: no SEAD was involved; only one company flew at a time; and CCSIL communications were strictly vertical (the company commander only communicated with his battalion commander and with his company through the medium of operations orders and situation reports.). The planner currently considers only one course of action and does not, as of yet, have the capability to evaluate the strengths and weakness of alternative courses of action. Another key limitation is that replanning capabilities are not fully implemented. It is likely that some details of execution and replanning will change as we gain more experience with these new capabilities.

We plan to expand the repertoire of behaviors available to the command entity and broaden the project to include higher levels of command. Currently, the commander plans for attack missions. We will soon broaden this to include missions of security and reconnaissance. These tasks appear to place more reliance on recognizing and adapting to changes in the situation, and we expect the explicit dependency information in our plan representation will be invaluable in providing these capabilities. By STOW97 we intend to have implemented a battalion level command entity.

7. Acknowledgement

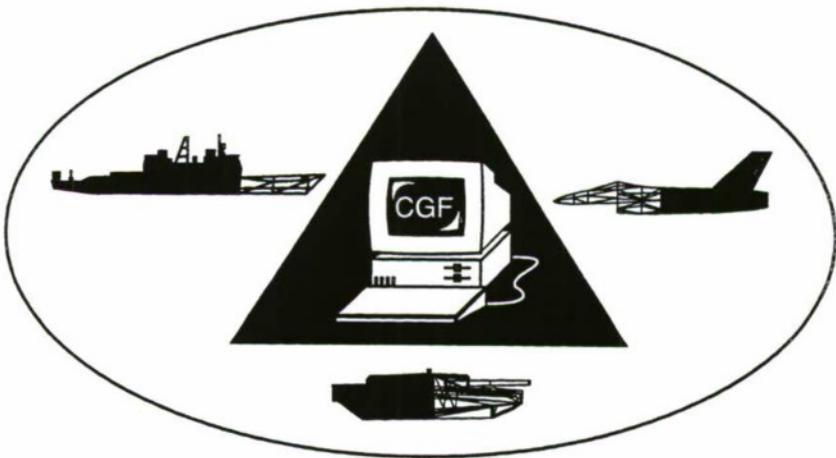
Jeff Rickel, Steve Chien, and Paul Rosenbloom assisted this work through their knowledge and valuable discussions. This research is supported under contract N66001-95-C-6013 from the Advanced Systems Technology Office (ASTO) of the Advanced Research Projects Agency (ARPA) and the Naval Command and Ocean Surveillance Center, RDT&E division (NRAD).

8. References

- Ambros-Ingerson, J. A. and Steel, S., "Integrating Planning, Execution, and Monitoring," *Proceedings of the National Conference on Artificial Intelligence*, 1988, pp 83-88.
- , D., "Planning for Conjunctive Goals," *Artificial Intelligence*, 32(3), 1987, pp. 333-378.
- Erol, K., Hendler, J., and Nau, D. S., "HTN Planning: Complexity and Expressivity," *Proceedings of the National Conference on Artificial Intelligence*, 1994, pp 1123-1128.
- Kambhampati, S., Knoblock, C. A., Yang, Q., "Planning as Refinement Search: A Unified Framework for Evaluating Design Tradeoffs in Partial_order Planning," *Artificial Intelligence*, 76(1-2), 1995, pp. 167-238.
- , M. "Planning with Constraints," *Artificial Intelligence* 16 (1981) pp 111-140.
- Laird, J. E., Johnson, W. L., Jones, R. M., Koss, F., Lehman, J. F., Nielson, P. E., Rosenbloom, P. S., Rubinoff, R., Schwamb, K., Tambe, M., van Lent, M., and Wray, R., "Simulated Intelligent Forces for Air: The Soar/IFOR project 1995," *Proceedings of the Fifth Conference on Computer Generated Forces and Behavior Representation*.
- McAllester, D., and Rosenblitt, D., "Systematic Nonlinear Planning," in *Proceedings of the Ninth National Conference on Artificial Intelligence*, 1991, pp. 634-639.
- , P., Johnson, W. L., Jones, R. M., Koss, F., Lehman, J. F., Nielson, P. E., Rubinoff, R., Schwamb, K., Tambe, M., "Intelligent Automated Agents for Tactical Air Simulation: A Progress Report," *Proceedings of the Conference on Computer Generated Forces and Behavioral Representation*, 1995.
- Sacerdoti, E. D., *A Structure for Plans and Behavior*, Elsevier_North Holland, 1977.
- M. R. Salisbury, M. R., Booker, L. B., Seidel, D. W., Dahman, J. S., "Implementation of Command Forces (CFOR) Simulation," *Proceedings of the Conference on Computer Generated Forces and Behavioral Representation*, 1995, pp. 423-432.
- , A., "Generating Project Networks," in *Proceedings of the International Joint Conference on Artificial Intelligence*, 1977.
- , D., "Domain_independent Planning: Representation and Plan Generation," in *Readings in Planning*, J. Allen, J. Hendler, A. Tate (eds.), Morgan Kaufman, 1990, pp. 319-335.

9. Author's Biography

Jonathan Gratch is a research computer scientist at the Information Sciences Institute, University of Southern California (USC), and a research assistant professor with the computer science department at USC. He completed his undergraduate education in computer science from the University of Texas at Austin in 1986. He received his Ph.D. in 1995 from the University of Illinois in Urbana-Champaign. His research interests are in the areas of planning, learning and decision theory.



Session 1b: Non-Military Uses of CGF

Adamson, DERA

Courtemanche, SAIC

Petty, UCF/IST

Miller, BMH



The CAEN Wargame for OOTW Applications

Janusz Adamson
CDA Land Studies Department
DERA, Fort Halstead,
Sevenoaks, Kent, TN14 7BP
United Kingdom

1. Abstract

The end of the Cold War has led to major changes in the world strategic environment which in turn have led to major revisions of NATO defence policies. Whilst the core role is still national defence, there is now a greater emphasis on the capabilities needed for regional conflict and on the requirement for Operations Other Than War (OOTW).

Wargames and simulations, which have been developed for defence applications, can provide excellent environments for the examination of paramilitary operations such as peace keeping and peace making (i.e. Operations Other Than War).

By peace keeping and peace making operations we mean such operations as operational planning, tactics, training, mission rehearsal, resource management, conflict resolution, crisis management and studying the complex decisions required for long and short term states of stability within the community.

2. Introduction

The Defence Evaluation and Research Agency (DERA) is a government owned research and technology organisation whose aim is to provide independent, high-quality, cost-effective scientific and technical services to its customers, primarily the Ministry of Defence (MOD).

The Centre for Defence Analysis (CDA), a division of DERA, provides advice and analysis of defence systems, procedures and operations, primarily to MOD. Operational Analysis (OA) forms a key element in the underpinning role of CDA in the decision making process for defence equipment procurement, defence planning and formulation of defence policy.

In order to meet these changing requirements, the CDA Land Studies Department at DERA, Fort Halstead, is enhancing a combination of wargames and simulations ranging from one-on-one to divisional and corps battles. One such development

is the Close Action ENvironment (CAEN), which can be run as either a wargame or a simulation. At the CDA, CAEN is used for operational analysis and weapon system evaluation. The UK Police may use CAEN for operational planning, tactics and training of officers in law enforcement and small arms situations.

3. CAEN

CAEN is a highly detailed model of the close combat battle. It is both a means of simulating weapons effects and an interactive wargame between opposing forces of up to platoon level strength. The area covered is typically 5-by-6 km and the terrain is represented at 10m resolution. Both urban and rural areas can be modelled with detailed representation of buildings and ground cover. Up to 200 entities are usually modelled and consist of either infantrymen and their personal weapons or vehicles such as armoured personnel carriers and main battle tanks.

CAEN can operate either as an automatically replicated simulation with no user intervention, or as an interactive game in which two or more independent players control the actions of their own forces.

The overall system comprises:

- Terrain editor facility.
- Interactive on-line gaming system.
- Deployment system.
- Game replay facility.
- Replication system.

3.1 Terrain Features

Terrain features include roads, rivers, minefields and obstacles. Vegetation can be represented as simple (a height and density) or complex (density varies with height) culture. CAEN represents complex culture as a number of different layers of varying density vegetation at different heights. Buildings are represented in higher resolution than other culture and include multiple storeys, sloping roofs and windows.

3.2 Interactive Gaming

The interactive system enables players to issue orders to allow them to change routes, arcs, activities etc. during the course of play. In this way the player can react to events in the game which are considered to be of military significance. The player does not have to control the actions of all the entities all of the time. Instead each entity or group of entities will follow an initial set of orders provided at the start of the game unless overridden by an interactive command. The wargame system also contains acquisition, movement, engagement and tactical models which are processed automatically.

The players can interrogate any entities under their control for relevant information such as damage status, ammunition remaining, etc.

The interactive gaming system is supported by a sophisticated colour graphics facility which allows each player a realistic view of the ongoing scenario, but constrained by the knowledge available to his own forces.

A save/restart facility allows players to save the state of the game at any time during play.

3.3 Movement

Movement is between nodes. Routes can be specified for groups or individual entities. Speed is limited by terrain. Infantry can change posture which may further limit speed (for example crawling). Infantry can be carried by vehicles and debussed from them.

3.4 Tactical Model

A major feature of CAEN is the tactical model which enables the player to set up tactics for entities or groups of entities. Objectives, aim zones and triggers can be set up to initiate or suspend activities (or behaviours) by entities or groups. Some triggers (for example suppression or cut wire) automatically initiate certain activities while others are set up by the player. Activities are either simple or complex. A complex sequence of activities is made up of simple activities which succeed or trigger one another. Group members may carry out different activities (for example fire and movement).

3.5 Detection

Detection occurs as follows:

- Random search detection methods use line-of-sight and target acquisition calculations; the models used depend upon whether the sight is optical, thermal or image intensifier.
- Detection using weapon signature is based on firing weapon, observer's sight and range.
- Detection is based on the noise made by the weapon being fired; it is also range dependent.

3.6 Engagements

Engagements are carried out using:

- Aimed fire. This is the same as direct fire in most models. Once a target(s) has been acquired, the entity will select a target.
- Suppressive fire. This is also direct fire, but it is directed at an area/object with the aim of suppressing any entities/objects in that area. It may incidentally cause casualties.
- Indirect fire. Missions are player directed or set up using aim zones and triggers.

3.7 Engagement Models

Engagement models include:

- Small arms. This model calculates the trajectories of rounds to determine casualties. Any hit by a bullet on a person is considered to have incapacitated the person.
- Fratricide. This models the accidental shooting of own troops.
- Explosive munitions. The mean area of effect is used to determine effects.
- Other munitions, for example armour piercing rounds. Hit probability and lethality are used.

3.8 Other Modelling Capabilities

Other effects of rounds modelled include:

- Suppression. This varies depending upon both the recipient and the incoming fire. Its effect is to cause the recipient to take cover.
- Obscuration (smoke) uses the COMBIC model. The smoke clouds drift across the map. They vary in size and shape, and depend primarily upon the type of round and meteorological conditions.
- Illumination. This modifies the (night) ambient luminance for a given area. The area will drift with the wind, and depends upon the ammunition type.

Minefields are defined by type and density of mine.

A “weapon sharing” model enables men to pick up key weapons whose operators have become casualties.

3.9 Fighting Within Buildings

Fighting within buildings, at the moment, is carried out using look-up tables. Entities of opposing sides enter combat once they are within a building and in the same 10m square. The fighting in buildings model then determines casualties with time. Entities can be added during the combat. Combat will continue until all of one side is killed or withdraws.

3.10 Other Facilities

A number of facilities are available to assist the player during planning the deployment. The most important of these is the ability to get a view of the battlefield from any given location (during the game the player can only get views from his entity’s current locations).

Day, night, and various meteorological conditions can be modelled.

4. Virtual Reality

Virtual Reality (VR) offers the capability to model and visualise 3-dimensional objects in real-time. A natural application of this enabling technology has been applied with great success to CAEN.

The VR facility within CAEN includes:

- A pre-game set-up facility. This allows players to move to any position and mimic entity viewpoints, thus helping validate line-of-sight assumptions and permitting a reconnaissance of the terrain.
- An interactive facility. This is achieved by networking and synchronising both gaming and VR environments. Thus, when a player displays a sensor view, the viewpoint data are transferred to the VR environment and the corresponding view displayed in the virtual world.
- A post-run analysis facility. This makes use of CAEN output to drive the virtual world. The analyst is able to move around the battlefield, view all static features, and monitor the unfolding battle from different perspectives without interacting with any of the entities. Alternatively, the analyst can clamp the viewpoint to a selected entity in order to check

for accuracy of modelling, orders and for desired behaviour

5. Operations Other Than War

Events which threaten life, property, the community or the environment, make particular demands, both financial and human, on those responsible for decision making when controlling and co-ordinating resources in response to those events.

The most critical period following such an event can be the time taken to respond, restore normality and take control of the situation, especially when initially there is incomplete information available.

This is especially the case where certain events have increasing economic, as well as national impact on the forces involved. Whilst organisational procedures, skills and drills exist, those with responsibility for decision making also need well developed human skills and knowledge in order to make them effective, even under stress. Operational officers (i.e. commanders) need to manage their doctrinal, organisational and leadership skills and then plan, exercise and test their theories in order to clearly see and understand how these situations evolve from initiation through to a successful achievement of the desired state.

To achieve this requires an approach which combines theory with practice. The theory can be encapsulated within products such as the Surrey Police Leadership Evaluation Action and Development (LEAD) product and the practice exercised and simulated within the CAEN environment.

Leadership training of operational officers, who are expected to take command and lead teams, is critical for mission accomplishment in operations such as:

- Terrorism.
- Hostage/Siege.
- Environmental disasters - fire, flood, etc.
- Explosions - natural, accidental, deliberate.
- Incidents - natural, accidental, deliberate.
- Industrial hazards.
- Public order, law enforcement, riot control.
- Movement control.

5.1 Modelling Methodology

A key requirement for modelling these events is the behavioural representation of neutral factions, non-combatants and crowds. This can be achieved by

aggregating numbers of people and representing them as single entities. For example, an entity may represent an individual trouble-maker within a crowd, whilst another entity may represent a group of people. Then taking advantage of the multi-screen, multi-sided capabilities within the wargame/simulation environment, it is possible to model and game a multi-national peace keeping force together with neutrals and non-combatants consisting of agitators, demonstrators, bystanders, the press, the police and any other interested party.

Other important components which need to be considered include how entities respond to noise, the realistic modelling of command and control (communications), the use of Command Agents, user interface issues, visualisation and object based (dynamic) terrain.

These components, when incorporated within a wargame or simulation, will provide a powerful generic environment for creating, testing and exercising plans in order to achieve a more positive outcome in the possible event of a major crisis or incident.

5.2 Behavioural Representation

Non-combatants are those entities who are present in the tactical area of interest but are not seeking to influence events. From a military point of view, these entities may represent refugees, evacuees, prisoners of war or members of non-military organisations. For non-military applications, they may represent the utility services, elements of a crowd, casualties, detainees, criminals, the news media or any other interested party. The emergency services, who are dealing with the crisis are the equivalent of combatants in a war game.

The presence, attitudes, activities and requirements of neutral factions, non-combatants and crowds can have a significant effect on the outcome of a major crisis or incident. If they are to be modelled, then the gaming/simulation environment must provide entity attributes such as:

- Disablement or injury, be it temporary or partial. This may include bodily functions such as sight, arms, legs, etc. The injury may affect mobility and posture.
- An entity may die from the wounds if not treated within a given time. Alternatively, the entity may be treated and recover from the injury.
- Effects of non-lethal weapons such as CS gas.

- Fear and aggression could be triggered dynamically by firearms, riot control equipment, agitators, noise, injury, etc. and would make an entity behave differently.
- Use of roman tactics during a scuffle, such as kicking and punching. Weapons such as boots, fists, batons and bricks would be made available.
- Dynamically changing clothing and weapons, such as the police having to change into riot gear, or the disarming of military personnel.
- The formation of wedges to break up crowds.
- Pushing activities such as aggregating individual entity strengths to move a vehicle.
- Making arrests, causing injury, detaining, removing or transferring injured entities from the tactical area.
- Individual entities could be identified and tagged for periods of time.

Within CAEN, these behavioural representations are activated by rules, triggers and data.

5.3 Multi-Screen

A multiple screen environment allows more than one player per side to participate in the interactive game sessions at the same time. By having several players per side, each side can be structured to represent several levels of command and control. This will allow investigations into communications problems between different levels of command.

Within CAEN each side can be divided into "command forces". A "commander" would be nominated for each side. Each commander may have additional players on his side so that more than one command force is represented for each side. The default would be one command force and one player per side.

5.4 Multi-Sided

A multi-sided system will allow more than two sides to participate in an engagement. This will allow non-combatant sides to be present, sides of unknown hostility, combat forces enforcing a cease fire or a UN peace keeping/peace making force. The multi-sided system will also allow for terrorists, criminal elements or different crowd factions to be represented.

CAEN allows each side to be deployed separately in the same manner detailed for separate forces in the multi-screen environment.

5.5 Noise

The modelling of noise and its effects on entities (noise may cause crowds to panic) needs to be considered. Noise may be broken down into background and foreground. Background noise is generally ignored, such as noise from lorries and cars, but the ambient level is significant. Foreground noise is generally sudden noise that may be heard above the ambient background; typical examples include a gunshot or a car backfiring.

5.6 Communications

Communications addresses the passing of information gathered by deployed entities throughout the chain of command. Current wargaming and simulation environments tend to have a perfect communications network. If communications are to be made more realistic, it is necessary that information is delayed, degraded or lost during the message passing process. Decisions, orders and actions will then have to be made upon imprecise information. Within CAEN, the communications network will exhibit the following characteristics:

- Errors in entity positions. The positions of entities on the screen will represent the last position at which a report was made to the operations officer. As more entities acquire a target, the target's position will tend towards its actual position.
- Delays in communication. Existing out of date information will be retained until an update is successfully communicated.
- Failure to communicate information. There may be situations where the information is not considered important.

5.7 Command Agents

Command Agents are used to represent decision making nodes within a command hierarchy. Each Command Agent represents a command post which is able to make decisions and interact with other Command Agents and entities within a wargaming, simulation environment. Command Agents therefore control operations within the tactical area of interest.

At the heart of a Command Agent is a knowledge based system containing explicit knowledge which describes sets of tactics and behaviours required by a command post to perform its particular role during the operation.

Facilities need to be provided for a human controller to take on the role of a Command Agent. This means that the human controller will perform all the decision making processes of that agent, thus replacing an existing agent or work independently alongside other agents within the command structure.

In a multi-screen, multi-sided environment, the use of Command Agents will reduce manning levels and hence the running cost of the simulated exercise.

Command Agents have been demonstrated with great success during a game at divisional level. These Command Agents are very sophisticated and contained a large number of rules. Individual entity Command Agents will not require very sophisticated knowledge bases.

5.8 User Interface

A human computer interface is that combination of physical components and software which combine to allow the user to issue commands to the computer, and allow the computer to present information to the user.

A user friendly interface is an essential requirement to ensure rapid acceptance of any system. A generic user interface is therefore required with a consistent look-and-feel for all application views. The user interface has to be intuitive, easy to use, reconfigurable and individually customised for specific applications. A graded "help" facility, which is activated by the user and based on the user's familiarity with the system, is a useful additional feature.

5.9 Visualisation

Virtual Reality (VR) provides an additional dimension to visualisation. The rapid creation of new terrain and objects, full immersive facilities for training people in leadership qualities under stressful/chaotic conditions, the smoothing out of movement between frames during gaming and replay, provide an important role in the decision making process.

5.10 Object Based Terrain

The requirement is to develop an overlay terrain structure so that terrain objects can be placed upon the terrain without being restricted by the grid structure of the current underlying terrain.

The proposed CAEN object terrain will allow a higher degree of detail to be represented in selected areas of interest. It will also provide a more realistic modelling environment for general terrain areas. This will provide the following benefits:

- Any size of terrain object will be possible, so a more realistic representation of features such as buildings, rubble, fox holes and trees can be modelled.
- The defining of terrain objects in terms of constituent elements will allow for a greater variation in the shape of buildings, trees, etc.
- The enabling of objects to be positioned on the terrain overlapping grid square boundaries will remove the uniformity observed in the existing terrain feature representation.

The current representation of a terrain area based on regular squares will be replaced by irregular triangles. This will provide a direct mapping to the VR implementation which uses flat irregular triangles for terrain representation. In the future, objects will be created within the VR environment and mapped directly into CAEN, thus considerably reducing the gaming set-up time for new geographical locations. However, additional processing may be required for terrain areas with complex contour detail.

6. Applications

Wargames and simulations, such as CAEN, can be customised to game and simulate threat management. By threat management we mean *“the positive management of any event which is a threat or potential threat to a state of stability”*. Public order operations, emergency planning operations and mission planning are examples of threat management.

The rapid creation of specific terrain and culture is an essential requirement for threat management. If data is not readily available, then a generic environment, such as a generic town, will suffice as an interim solution. This town might contain a railway station, town centre, county court, police station, sports town by-pass.

A typical operation might address the tactics and resources required to police elements of a crowd moving from one part of a town to another. Police cars would be used to shepherd people along main roads whilst additional police cars and police officers would also be allocated to strategic positions to cordon off parts of the town. Potential application areas include the control of football crowds, the

policing of the annual carnivals or the containment of riots following an unpopular event or decision.

Another operation might be concerned with the co-ordination of the emergency services following a major incident in a built-up area.

A typical incident report may read as follows:

SCENARIO

*Date: 20 June 1996
Time: 09.30 am
Weather: Overcast - outlook rain
Wind direction: NNE speed 5 mph*

INCIDENT

At 09.30 am a fully laden tanker, travelling EAST through the village of Copehill has overturned whilst manoeuvring around a tight corner. Immediately behind the tanker is a group of foreign tourists in a car.

The tanker explodes and the tanker driver is killed. The driver of the tourist car is killed and one of his party is injured.

The person in the house next to where the tanker explodes tries to help the driver, but is killed in the attempt.

The remainder of the family in the house have walking injuries and are able to move to a safe location, but do require urgent medical assistance.

Wargames and simulations such as CAEN can be used to create the above incident and encourage operations officers to take control of the situation and restore normality within a constructive, virtual environment.

It has been observed that the most critical period following any incident can be the time taken to respond, restore normality and take control of the situation at the scene of the incident, especially when initially there is incomplete information available. Typical questions that may need to be addressed are:

- How quickly can the emergency services arrive at the scene of the incident?
- What is the quickest route to the incident?
- What resources are required to restore order and take control of the situation?
- What measures are required to prevent escalation?
- Where are the most suitable locations for establishing command posts?

7. Safety Management

Analysis of the major incidents reported within the European Community indicates that, in the majority of cases, management error was the underlying cause. This error can manifest itself as deficiencies of organisation, inadequate training, or simply failing to take into account the possibility of human error.

The cost of running live exercising in order to minimise these errors is excessive. For example, when Eurotunnel decides to close down one of their railway tracks for such a live exercise, the loss of revenue could be excessive especially when trains are scheduled to run through the Channel Tunnel at 3 minute intervals.

The cost of such live exercises, excluding manpower costs, may range from £15K to in excess of £150K (for example, a 3 hour exercise at a provincial airfield costs in the region of £100K). When the Control Of Major Accident Hazards (COMAH) legislation becomes law, the number and frequency of running different types of live exercises may increase.

7.1 COMAH

COMAH is a major European accident prevention policy which will set out in writing an operator's aims and principles for the control of major hazards in an establishment, and in particular, the safety management system which is controlled by that operations officer.

Operations officers will have to prepare emergency plans and explain how they will respond should a major incident occur. They will have to provide sufficient information to the authorities to enable them to draw up off-site emergency plans. Part of the plan will be the requirement to inform the public within the vicinity of the incident what actions should be taken.

Wargames and simulations such as CAEN could help validate the quality risk assessment procedures required when the COMAH legislation becomes law.

7.2 Safety Exercises

The testing of an emergency plan may prove to be pointless as it has been observed that these exercises tend to be repetitive and may not necessarily test the critical components of the plan. Existing wargames and simulations could provide the solution.

The real business behind training and exercises is to test the foundations of corporate, organisational and personal responsibility. Distributed Interactive Simulation (DIS) techniques and protocols would provide a multi-agency approach to the planning, exercising and testing of emergency plans. A distributed system would consist of workstations which are located at various sites and linked together over a wide area network. This system architecture would enable operational officers, who are sited at various geographical locations, to communicate, game, simulate and exercise their plans together in a realistic virtual environment.

8. Conclusion

Wargames and simulations, such as CAEN, provide excellent environments for gaming and simulating para-military operations such as:

- Operational Analysis (OA) on such topics as Close Combat, Military Operations in Built-up Areas (MOBA) and Key Point Defence.
- Operational planning and training.
- Reviewing command and control.
- Evaluating human performance.
- Training personnel for given emergencies.

The flexibility of CAEN and the very fine detail of its modelling capability provide an excellent environment for Operations Other Than War. Within reason, anything that moves on land can be modelled and gamed within CAEN. This is illustrated by the ease with which CAEN has been used for creating a variety of different scenarios for both defence and non-defence applications.

There are many potential uses of CAEN for peace keeping, peace making and paramilitary activities. These include operational planning, tactics, training, mission rehearsal, resource management, conflict resolution, crisis management and studying the complex decisions required for long and short term states of stability within the community.

The benefits of using wargames such as CAEN include:

- A distributed computer environment to visualise, interact with and rapidly re-configure complex events and disorder.
- The ability to plan, practice and test a variety of responses to emergency and critical situations in a tailored environment.
- A facility for pre-operational and post-operational analysis on such topics as threat

analysis, vulnerability analysis and risk assessment.

- A substantial reduction in the costs incurred when an organisation sets up “real” situations to simulate complex events and disorder.
- Reduction in the costs and damage to reputation which occur when organisations “get it wrong” in an emergency and find that they are the victims of damage litigation.

In summary, wargames like CAEN offer forums to explore the synergy between the terrain, the environment, and the man-in-the-loop for both defence and non-defence applications.

9. Author’s Biography

Janusz Adamson is a Senior Consultant at the Centre for Defence Analysis, DERA Fort Halstead. Mr. Adamson has a BSc(Hons) degree in Astronomy and an Mphil. His project responsibilities include (CASUM), the Close Action Environment wargame (CAEN), Genetic Algorithms, Real-time Knowledge base Systems and Command Agents. His technical focus is on Synthetic Environments and Computer Generated Forces.

MedSAF: Prototyping a Vision for Medical Simulation in DIS

Anthony J. Courtemanche and Kent Bimson, Ph.D.
Science Applications International Corporation
3045 Technology Parkway
Orlando, FL 32826-3299

Anthony_Courtemanche@cpqm.saic.com
Kent_Bimson@cpqm.saic.com

1. Abstract

Dramatic improvements are needed to increase the level of medical readiness in the Department of Defense, FEMA, and other medical services. These improvements can be achieved through innovative application of simulation technology. We have developed a vision for simulation support to medical readiness based on medical extensions to, and linkages among virtual and constructive combat simulations, command and control systems, advanced patient MIS systems, and medical training simulators.

As a first step in implementing our vision, we have developed a prototype Medical Semi-Automated Forces (MedSAF) system based on medical extensions to the Modular Semi-Automated Forces (ModSAF) combat simulation system. We have also developed a modem-line linkage between MedSAF and the Human Patient Simulator (HPS), a scenario-based, parameter driven mannequin-style simulator developed by the University of Florida School of Medicine. We can demonstrate a fully integrated medical scenario that includes combat, generation of infantry casualties, simulation of first care treatment, evacuation to higher echelons of care via combat ambulances and evacuation helicopters, and models of treatment at a Battalion Aid Station and Evacuation Hospital. Casualty models have been developed that change state over time, including vital sign degradation (e.g. pulse, blood pressure, blood loss) based upon casualty type.

In this paper, we describe the design and implementation of these MedSAF extensions to ModSAF. This publication is a follow-on to our previous report and it updates the status of our development since that publication and provides more detail relevant to the DIS and CGF communities.

2. Introduction: The Need to Improve Medical Readiness

As we have reported in our previous paper (Courtemanche *et al.* 1996), the tri-service medical community is currently focused on meeting the medical readiness challenges imposed upon it by the

digitized battlefield requirements, including training, mission rehearsal, leadership development, doctrine evaluation, materiel solutions, and the need to test and evaluate a system's readiness for fielding. The medical community is attempting to accomplish these goals in the face of decreasing budgets and increasing technical, personnel, medical, and threat challenges.

In the past, many of these challenges have been met through direct training on live systems and through live simulation exercises, approaches that are becoming increasingly expensive and which suffer from the liability that they are non-repeatable, uncontrollable, and, in many cases, medically deficient. As has been amply demonstrated for combat forces, simulation offers key technology to help meet many of these challenges. Unfortunately little has been done to date to support medical readiness through the use of Advanced Distributed Simulation (ADS), and this has prompted the development of our vision for improving medical readiness through simulation linkages.

3. Improved Medical Readiness Through Simulation Linkages

SAIC's ASSET and Health Care Technology Group organizations have put together a vision for how the goals of medical readiness can be achieved by extending current simulation systems to play medical processes, and then linking them to live medical equipment to support military medical training, system evaluation, and procedure validation.

The key components of this vision are:

1. linking different types of simulations,
2. exploiting the synergy provided by such linkages, and,
3. extending the systems with medical play.

Sections 3.1 through 3.5 describe the types of simulations we envision linking together to provide a superior solution to the challenges of medical readiness.

3.1 Virtual Simulations

An example of virtual simulators are Semi-Automated Forces (SAF) applications and the crewed simulators that these applications interact with. One such example is ModSAF (Courtemanche & Ceranowicz 1995). ModSAF, or Modular Semi-Automated Forces, is a Computer Generated Forces (CGF) system that researchers can build upon and extend. It is fully compatible with DIS network protocols. Its development has been funded by the Defense Advanced Research Projects Agency (DARPA) and the Army's Simulation Training and Instrumentation Command (STRICOM). The latest version, ModSAF 2.1, was released in May 1996, and it contains over 750 thousand lines of software written in C.

One of the current users of ModSAF is DARPA's Synthetic Theater of War (STOW) program. STOW has the objective of demonstrating the use of ADS for large scale exercises at the Joint Task Force level distributed over many sites, including linkages to constructive simulations and live players (Aronson 1996). The STOW program is currently enhancing ModSAF in the areas of service-specific synthetic forces, synthetic environments, and simulation networking, leading to the STOW '97 training exercise.

3.2 Command & Control Systems

An example of Command and Control (C2) systems is the Army's Phoenix system, formerly know as the Battle Command Decision Support System (BCDSS). Phoenix, developed by Mystech Associates, is a real world command and control system that allows commanders to organize, analyze, display, and manipulate information about their forces on the battlefield. It is one of a group of systems that are attempting to revolutionize the way in which command decisions are made and combat data is disseminated. Phoenix is not a simulation, but is used extensively in training exercises and is integrated with existing training simulations. The system provides a relational database, tactical maps, communications tools, decision support tools, and command matrices on a computer. The computer system increases the speed at which information can be generated, exchanged, and understood by commanders. As such, it is a force multiplier, making U.S. forces more responsive and effective against their enemies.

3.3 Advanced MIS Systems

An example of an advanced medical MIS system is the Trauma Care Information Management System (TCIMS). Under sponsorship by DARPA, TCIMS

is being developed by various consortium members as the next generation medical MIS system for the DoD. It provides unprecedented levels of accurate patient information to various echelons of care, starting at level 1 (medics treating individual soldiers at incident sites) up to levels 3 and 4 (military hospitals treating large groups of casualties).

3.4 Patient Simulation

An example of patient simulation is the Medical Education Technologies Inc./University of Florida Human Patient Simulator (HPS). The HPS is a full scale, life-like simulator that is model-driven and script controlled (Lampotang *et al.* 1995 and van Meurs *et al.* 1993). This hybrid system allows users to optimally and creatively take advantage of both types of control. The cardiovascular features of the HPS include palpable radial and carotid artery pulsations, heart sounds (normal and abnormal), 5-lead electrocardiogram, non-invasive blood pressure measurements, and invasive arterial, central venous, pulmonary artery, and wedge blood pressure. All these measurements are made using standard monitoring equipment.

Simulation scenarios can be constructed for individual patients. This allows for the implementation of specific script driven events (e.g. a certain amount of blood loss from an injury). The physiologic data will respond to those events in a realistic manner as dictated by the physiologic model.

Scenarios including combat casualties such as wounds causing blood loss, pneumothorax, and insufficient oxygen uptake caused by chemical weapons or smoke inhalation are possible. Development of the HPS is still ongoing and currently work is underway to add brain, eye, and neurological features to the HPS. The modular design of the HPS and the ability to program different patients via the scenario editor makes the HPS the ideal human model to be used in combat simulation.

3.5 Simulation Linkages for Medical Readiness

Each of the above systems excels in its respective domain. The ModSAF combat simulation provides valid representations of combat activity and can populate a virtual battlefield with large numbers of simulated entities. Phoenix, as part of the Army's Maneuver Control System (MCS), can receive, manipulate, and send a wide variety of command and control messages, including reports and orders. TCIMS holds the promise of dramatic improvements in collecting, maintaining and retrieving accurate patient data. The HPS provides a simulation and training environment that allows medical practitioners

to practice medical procedures using actual medical equipment.

Our vision for linking the above simulations and systems to improve medical readiness can be summarized in Figure 1.

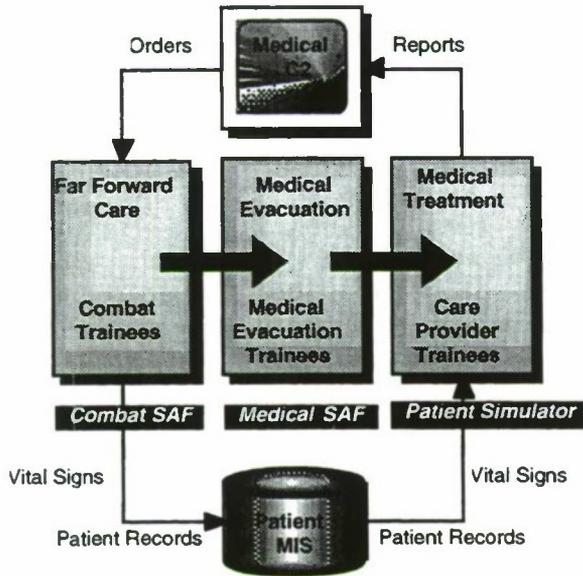


Figure 1: Simulation Linkages

We propose development of a medical training system that includes representation of the combat mission. As casualties are generated in combat SAF simulations such as ModSAF, these casualties can populate patient MIS systems. Given a Medical SAF capability that can accurately model the medical logistics as well as maintain a suitable representation of casualties and their treatment at the multiple echelons of care, training of medical evacuation logistics can be performed. When linked to patient simulators such as the HPS, training for care providers can be accomplished in the same scenario. Properly linked to the advanced patient MIS as well as actual command and control equipment, a seamless training scenario that exercises all aspects of the medical mission is possible.

The benefits of this proposed linkage are that it allows multiple uses of individual simulations in a combined fashion. Simultaneous training at varying resolutions and levels of care becomes possible.

Since doctrine dictates that the medical mission must support the combat mission, we stress that it is crucial to extend accepted combat simulations with medical play rather than to develop them in stand-alone mode.

4. The MedSAF Prototype

As a first step in implementing our vision, we have developed a prototype Medical Semi-Automated Forces (MedSAF) system based on medical extensions to ModSAF. MedSAF was then integrated to interoperate with the HPS. We call the combined MedSAF and its linkage to the HPS, MedSIM (Medical Simulator).

The remainder of this paper describes the capabilities and implementation of MedSAF and linkage to the HPS. This integrated MedSIM capability can be used to demonstrate a fully integrated medical scenario that includes combat, generation of infantry casualties, simulation of first care treatment, evacuation to higher echelons of care via combat ambulances and evacuation helicopters, and models of treatment at a Battalion Aid Station and an evacuation hospital. Casualty models have been developed that change state over time, including vital sign degradation (e.g. pulse, blood pressure, blood loss) based upon casualty type.

A remote linkage to the Human Patient Simulator allows substitution of a life-like simulator for MedSAF casualties, just as tank simulators may be substituted for ModSAF tank entities in a distributed simulation. Much as virtual tank simulators allow platoon leaders and tank crewmen to train in the combat context represented by ModSAF, the HPS allows medical professionals to train on a human-like simulator in the combat context represented by MedSAF. The HPS provides a powerful environment for training in triage and treatment of casualties throughout the course of the simulation. Our prototype system can be made fully compatible with existing DoD standards for DIS or DMSO's emerging High Level Architecture (HLA).

4.1 Extensions to Support Medical Simulation

The extensions developed to produce a medically credible MedSAF from the ModSAF combat simulation system are described below. These extensions were specifically developed to support the execution of the demonstration scenario described in section 5.1

4.1.1 Medical Support Vehicles

The first development task under the MedSAF project was to ensure that the proper entities existed to populate the synthetic battlefield in the demonstration scenario. Refinement of the scenario revealed the requirements for a M113 combat ambulance and an evacuation helicopter, as described in the following

sections. In addition, a generic individual combatant, enhanced to support casualty modeling was created.

4.1.1.1 M113 Combat Ambulance

ModSAF already contained a baseline version of a M113 ambulance; however no medical modeling capabilities or patient transportation were available for that vehicle. The MedSAF project enhanced the baseline ModSAF M113 ambulance by adding the specific transportation behaviors described later in the paper. This was easily accomplished by updating the M113 ambulance configuration files to include the capabilities to execute the specific unit level task described in section 4.1.2

4.1.1.2 UH-53 Evacuation Helicopter

To implement air evacuation from the Battalion Aid Station (BAS) to the evacuation hospital, a specific UH-53 air evacuation helicopter was created in MedSAF. This was accomplished by creating a parameter file for the UH-53. This helicopter was extended with transportation behaviors in the same manner as the M113 combat ambulance.

4.1.1.3 Dismounted Infantry

A generic infantry entity was created (again using ModSAF's ability to define new entities via data files) to implement casualty generation and modeling of wounded patients. For the purposes of the demonstration, this infantry had no specific weapons. This infantry entity was extended with the casualty generation algorithm, casualty transportation and casualty representation modeling described in the following sections.

4.1.2 Casualty Transportation

To support the transportation of wounded casualties across the battlefield, the ModSAF behaviors that already allow dismounted infantry to mount vehicles were investigated. An analysis of the existing ModSAF mount and dismount behaviors revealed serious limitations that would restrict the ability for combat ambulances to transport arbitrary casualties to and from arbitrary echelons. At the time, ModSAF's capabilities to allow soldiers to mount and dismount vehicles and to be transported across the battlefield were limited to infantry that were task organized as part of integrated vehicle/individual combatant combat units. In this form, this implementation would unacceptably limit the transportation of casualties. For example, the ability to transport enemy or non-aligned casualties would not be supported.

A design for flexible "mounting" and "dismounting" of wounded from ambulance vehicles (both the M113 combat ambulance and the UH-53 evacuation helicopter) was developed to overcome the ModSAF shortcomings. This design relies on message passing

between the ambulance vehicle and the patient to coordinate the pickup and delivery of the wounded, as shown in Figure 2. This message passing is accomplished via encoded DIS Signal Protocol Data Units (PDU's) that allow different entities in a networked simulation to communicate with each other.

The contents of the "Board Me" message indicate the DIS entity ID of infantry being commanded to board the ambulance, and the entity id of the ambulance to board. The contents of the "Boarded" message provides a positive acknowledgment that the boarding occurred, also by indicating the respective entity ID's. The implementation of the "DeBoard" and "DeBoarded" messages are similar.

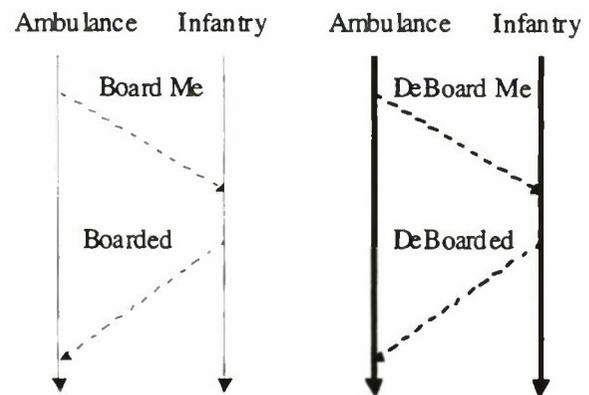


Figure 2: Messages and Timelines for Casualty Pickup and Delivery

This approach of using DIS messages to communicate and acknowledge the boarding and deboarding process allows potential future interoperability with the boarding and deboarding of non-CGF individual combatants onto non-CGF vehicles.

The boarding capabilities were implemented by a pair of ModSAF tasks, as described below.

4.1.2.1 Vehicle Board

A ModSAF vehicle level behavior called VBoard was created for the wounded infantry to monitor requests from ambulances to be picked up or dropped off. As in all vehicle level ModSAF behaviors, this behavior has direct access to the vehicle it supports. This access enables the behavior to directly cause the entity to deactivate (leave the DIS exercise) or reactive (rejoin the DIS exercise).

Based loosely on the baseline behaviors for the mounting and dismounting of DI Groups, VBoard continually examines incoming radio messages for requests to be picked up, or, if already boarded, to be

dropped off. If a request from an ambulance to board is received, the infantry deactivates itself, leaving the DIS exercise, and it becomes dormant. When a request from the ambulance to deboard is received, the infantry entity reactivates at a location near the ambulance vehicle. This gives the appearance that as the vehicle moves, the boarded infantry has moved along with it.

4.1.2.2 Unit Evacuation

A unit level behavior, UEvac, was created for transportation vehicles to initiate the request to pickup or deliver wounded infantry. This behavior was based loosely on the behavior to pick up DI Groups. As a unit level behavior, it can be directly assigned to a unit or vehicle from the ModSAF GUI. The operator assigns an Evacuation mission to either an ambulance or Medevac helicopter. The operator must supply a location at which to perform the evacuation or delivery. Depending on the type of vehicle (ground or RWA), the UEvac behavior (task) that is part of the Evacuation mission (taskframe) spawns the appropriate movement sub-behaviors to move the vehicle to the evacuation or delivery point.

Once the vehicle arrives at its destination, if the operator has configured the mission for pickup, the vehicle searches an operator-controlled search radius for wounded infantry. Once located, "Board Me" messages are sent to the wounded infantry. If the mission has instead been configured for delivery of the wounded, the behavior instead sends "DeBoard Me" messages to all the infantry that have previously boarded. Receipt of "Boarded" or "DeBoarded" messages allows the unit behavior to positively track the current number of infantry it is carrying.

The fact that the behavior is sensitive to whether it is running on a ground vehicle or RWA vehicle (and that it determines which particular movement sub-behaviors to invoke based on vehicle type) is a unique design in the domain of ModSAF behaviors. It is possible and desirable in this case because evacuation by air or ground is extremely similar at this resolution of modeling.

4.1.3 Casualty Representation

The most significant development activity in the MedSAF project was the development of credible medical models. This was accomplished via the development of a flexible modeling language, and the use of subject matter experts to help develop models within that language, as described in sections 4.1.3.1 through 4.1.3.3.

4.1.3.1 State Interpreter

In the MedSAF project, we developed a flexible interpreted computer language called the State

Interpreter Language (SIL). The high level purpose of SIL is to represent physical and behavioral models in a data driven interpreted fashion to facilitate development and debugging. SIL was specifically used in this project to model the evolution of human casualty states.

SIL allows the declaration of state variables of integer, floating-point and string types. These variables can be initialized and updated through the actions of a full suite of mathematical operators. Through the use of defined states and control-flow statements, conditional expressions can be defined. The basic SIL language is interpreted via a SIL interpreter. The SIL interpreter can run a SIL program stand-alone as well as imbedded within MedSAF. The SIL interpreter has been designed to allow easy extension of the SIL language through the registration of named primitives. External applications can interface with SIL through shared variables or primitive extensions.

The advantage of the SIL language for the implementation of medical models and behaviors is that it allows the rapid addition of models and behaviors into the MedSAF system without recompilation. For example, we were able to easily add new medical sub-models without code compilation, such as a model of diastolic and systolic blood pressure derived as a function of mean arterial pressure (MAP). This facilitates development and explorations into new behaviors or models.

4.1.3.2 Human Physiological Modeling

In MedSAF, we used SIL to implement several prototype medical models to represent the time-evolving state of human casualties. The use of SIL in MedSAF was dictated by the requirements to easily create MedSAF models. Given that the proper state variables that define a human casualty were unknown until late in the development cycle of the project, it was essential to develop a system that facilitates model development without compilation. As the models become more refined and gain stability in implementation, these models can be implemented in ModSAF's compiled Finite State Machine (FSM) language (Calder *et al.* 1993), for efficiency of execution.

Our current medical model was created with medical subject matter advice provided by medical modeling experts from the University of Florida Department of Anesthesiology. The model represents a human patient via several coupled sub-models, which include a cardiovascular model and a model of blood oxygenation. A model of brain death based on blood-pressure and blood oxygenation determines the health of the patient.

4.1.3.3 Treatment Modeling

In order to feed medical treatment inputs to the MedSAF human physiological models, the SIL language had to be extended with primitives to determine if the patient has a healthy buddy available to treat wounds, whether the patient was in an ambulance, whether the patient is at the Battalion Aid Station, or whether the patient is at the evacuation hospital. Because of SIL's ability to accept named extensions via code registration, these extensions were straightforward to implement. Each of the primitives used ModSAF search primitives to determine the nearness of other DIS entities (healthy buddies) or certain graphical objects (BAS or evacuation hospital). A primitive to determine whether or not the infantryman is in an ambulance was also added.

Based on the results returned from these primitive extensions, the SIL-encoded casualty model will dispatch to appropriate treatment logic, such as a change in blood oxygenation due to intubation and ventilation. In this manner, we have demonstrated how to provide "echelons of care" to simulated casualties within a DIS combat simulation.

4.1.4 Casualty Generation

Prior to the transportation and treatment of casualties in MedSAF, casualties must be generated as a result of combat. In support of the development of high-resolution casualty generation in MedSAF, a limited survey of casualty generation data sources was performed. The results of the limited survey were disappointing, in that the only available documented models deal with human injuries at a very high level, consistent with the aggregate vehicle-level damage states of mobility kill, firepower kill, and catastrophic kill. Clearly this level of injury representation was too coarse to be used in a medical scenario. Recent contacts within the Army medical community indicate that higher resolution human injury models are available, and these will be examined as part of follow-on MedSAF development.

As a result of the limited injury-modeling information available at the time of development, a new ModSAF damage library was created. This library generates injury events as a result of direct or indirect fire, according to the datafiles and algorithms described in sections 4.1.4.1 and 4.1.4.2.

4.1.4.1 Direct Fire Casualty Generation

Taking ModSAF's direct fire model as an example (Courtemanche & Monday 1994), the following data structure was used in the casualty library to determine whether an infantryman has sustained an injury due to a small-arms direct fire event.

```
(  
  ("No Injury"      0.05)  
  ("Severed Limb"  0.05)  
  ("Flesh Wound"   0.40)  
  ("Chest Wound"   0.30)  
  ("Head Wound"    0.10)  
  ("Death"         0.10)  
)
```

This data specifies that there is a 5% chance of "No Injury", 40% chance of "Flesh Wound", etc. The probabilities must add up to 1.0. A different set of probabilities can be associated with different targets and different weapons impacting the target. Being data driven, the statistics can be easily changed to correlate with empirical data.

The benefit of this data representation is that the damage events are completely data-driven. For example, a new damage event such as "Abdominal Wound" can be added to the data file without recompilation. As these damage events are routed directly to the SIL-encoded human physiological models, completely new damage events and resulting human patient outcomes can be created just by augmenting data files.

4.1.4.2 Indirect Fire Casualty Generation

For indirect fire (that is weapons impacts directed at a location as opposed to a specific vehicle), a slightly modified damage file is used:

```
(  
  (0.0 5.0 ((("Death"      1.0)))  
  (5.0 10.0 ((("Head Wound" 0.1)  
              ("Death"     0.9)))  
  (10.0 25.0 ((("Head Wound" 0.1)  
               ("Chest Wound" 0.1)  
               ("Death"     0.8)))  
  (25.0 50.0 ((("Head Wound" 0.1)  
               ("Chest Wound" 0.1)  
               ("Flesh Wound" 0.2)  
               ("No Injury"   0.1)  
               ("Death"     0.5)))  
  (50.0 100.0 ((("Chest Wound" 0.1)  
                ("Flesh Wound" 0.2)  
                ("No Injury"   0.7)))  
)
```

This data file contains damage probabilities as in the direct fire case; however these probabilities are associated with different range bands. For example, between 0 and 5 meters away from the indirect fire impact, this data file indicates 100% chance of "Death".

4.2 Linkage to the HPS

MedSAF was envisioned not as a standalone system, but as a system capable of being networked with other simulations to provide multi-level medical combat training. An example of this is the demonstrated linkage of MedSAF with the HPS, which was prototyped under the HPS Pilot Study, as described below.

The overall concept of linking MedSAF and the HPS is based on current ADS training environments, in which manned combat simulators (i.e., tanks, APC's, etc.) are linked to SAF combat forces. Likewise, medical combat training requires a high resolution manned simulation interface (the HPS), as well as semi-automated forces (MedSAF). The simulator portion (i.e., HPS) can be used to model certain MedSAF casualties at a sufficiently high resolution that effective "team training" (i.e., training of medical practitioners) can take place.

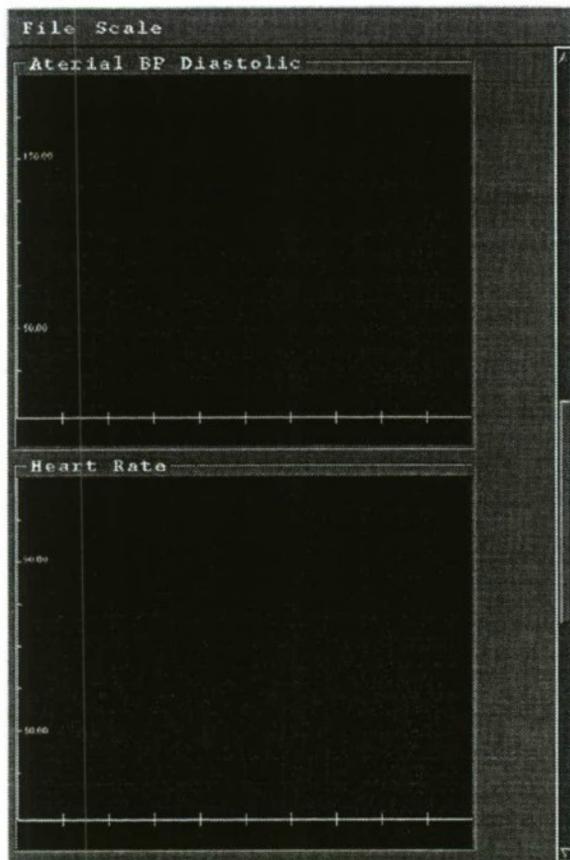


Figure 3: MedSAF Display of HPS Data

4.2.1 Human Patient Simulator Pilot Study

In the HPS Pilot Study, we prototyped a linkage between the HPS and MedSAF. Under the Pilot Study, modem-based serial communications was used

to communicate patient data from the HPS in Gainesville, Florida to MedSAF running in Orlando, Florida. Due to the limited schedule for the Pilot Study, a very rudimentary linkage was designed and implemented. The HPS communicates scalar (non-waveform) data, such as breath-rate, via the modem link. A standalone communications program collocated with the MedSAF system receives the data from the HPS and displays it graphically over time, as in Figure 3.

The current linkage is two-way in that a standalone program communicates commands to the HPS to initiate the linkage, load up the patient with a pre-planned patient scenario, and receive vital sign data back from the HPS. However, the linkage is limited in that the particular patient configuration is pre-planned and not a reflection of the particular state of a particular casualty in MedSAF. This extension is planned in the next version of MedSAF.

4.2.1.1 MedCOM Serial Communications

Serial modem communications between the HPS and MedSAF was accomplished using a reliable communications protocol specifically designed to transmit physiologic data over modem connections. This protocol, named MedCOM, uses data packets of variable length, a checksum and a positive acknowledge algorithm for reliability. This protocol was tested with data generated by the HPS over a telephone link between Gainesville, Florida and Orlando, Florida. Link interruption tests show that the protocol is reliable and that it has the ability to resynchronize a connection when synchronization is lost.

Although the normal linkage to the HPS is currently over a modem line, we have demonstrated the ability to maintain a direct serial line linkage in the case where the HPS and the MedSAF workstations are collocated. Also, we believe that the MedCOM protocol can be easily adapted to DIS via the encapsulation within Signal PDU's or the use of the Simulation Management Protocol. In the future, as HLA implementations become prevalent, the MedCOM communications model can easily be converted to a Run Time Infrastructure (RTI) which can communicate changing attribute values (vital signs) with specified reliability characteristics.

4.2.1.2 Remote Interface to HPS

To facilitate remote control and remote data collection from the HPS, the HPS software has the ability to run one or more remote controls that are connected to one of the multiplexer serial ports contained within the HPS. To connect the HPS to the MedSAF communications program and data display, a gateway

was designed that communicates as a remote control to the HPS and on the other side connects to a modem and transmits the data that the MedSAF communications program wants to receive. All communication is via the MedCOM protocol. The HPS-MedCOM gateway software requests physiologic data from the HPS every five seconds and sends it on via the modem with the MedCOM protocol. Commands coming from the MedSAF communications program are translated and validated and sent on to the HPS. An example of a command is "start patient", which will cause the HPS to start loading a specific patient scenario.

4.2.1.3 Data Grapher

In order to display real-time remote patient state from the HPS to the MedSAF operator, an X-Windows and Motif based display subsystem was created to graphically display time-changing MedSAF and HPS data. This system, called Data Grapher, can plot multiple synchronized waveforms, as in an EKG/respiration monitor. This display system was integrated with the standalone MedSAF HPS communications program to plot returning HPS variables, as in Figure 3. In the future, this display system will be integrated directly into MedSAF to plot the state variables of the low resolution MedSAF medical models.

5. Project Status and Results

The MedSAF and HPS Pilot Study were sponsored by SAIC's Independent Research and Development program. Successful demonstration of the capabilities described in this paper was given in the first quarter of 1996. A fully integrated scenario including combat, generation of casualties, treatment of the casualties at different echelons of care, transportation of casualties to different echelons of care, and linkage to the HPS has been demonstrated and briefed to several representatives of the Department of Defense Simulation, Training, and Medical communities. This demonstration scenario is described below.

5.1 MedSAF Scenario

A mech infantry platoon, part of a Mech Infantry Company Team, is attacked, and casualties are sustained to a dismounted infantry squad. First care to the wounded is provided by an M113A3 Combat Ambulance, which had already been task organized to the Company Team from the Battalion Medical Platoon. The Combat Ambulance moves forward, from its normal position with the Company trains 1000 meters behind the front line of the Company team, to assist the wounded squad. The casualties are transported by combat ambulance rearward to a patient collection point behind the Company Team defenses.

Unable to treat all the casualties, the Company 1st sergeant requests ground evacuation of the wounded to the Battalion Aid Station (BAS). The BAS dispatches one of its un-tasked M113A3 ambulances to retrieve and transport the wounded from the Company to the BAS.

At the BAS, care is given to the retrieved wounded by the Battalion surgeon. A critically wounded infantryman with chest trauma is stabilized prior to evacuation by air via UH-53 medical evacuation helicopters. The infantryman is transported to the Evacuation Hospital for treatment, including intubation for general anesthesia during chest surgery.

Figure 4 below is a graphical representation of the scenario.

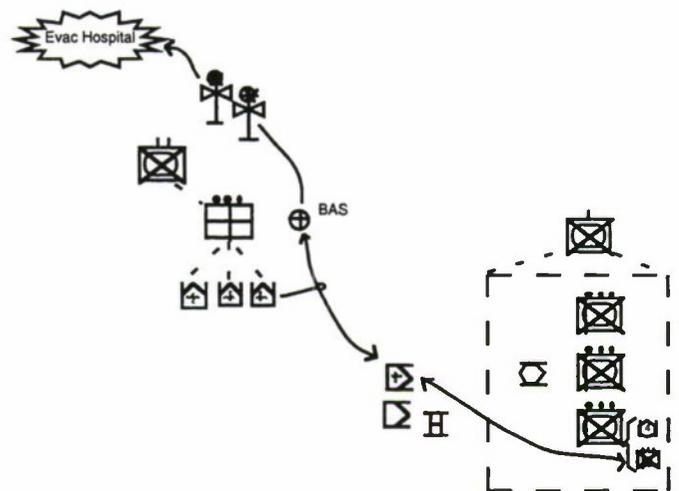


Figure 4: MedSAF Scenario

5.2 MedSAF Scenario Execution

During the course of the execution of this scenario, all of the capabilities of MedSAF and the linkage to the HPS are exercised. This is graphically depicted in Figure 5.

The basic capabilities of ModSAF are used to lay down the forces for this scenario, including the dismounted infantry, combat vehicles, medical platoon, evacuation helicopters, and map annotations (graphical Persistent Objects) to represent the Battalion Aid Station and the evacuation hospital.

When the infantry come under indirect fire, the MedSAF casualty generation algorithms dynamically produce a statistical distribution of casualty types. These casualties execute the MedSAF patient and treatment models. For example, an injured

infantryman can receive positive treatment for certain medical problems if a healthy infantryman is nearby to provide aid.

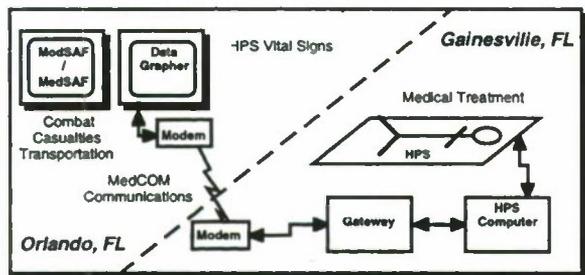


Figure 5: MedSAF Architecture

The MedSAF operator chooses where and when to dispatch the Company and Battalion ambulances to evacuate casualties. Using the MedSAF patient transportation capabilities, casualties are moved to higher echelons of care, including the Battalion Aid Station or the evacuation hospital. All the while, the MedSAF patient and treatment models execute, reacting to the types of treatment available in the different echelons of care. If the casualty is severe enough, or if the operator fails to evacuate the casualty fast enough, the patient may lapse into unconsciousness, sustain irreversible brain damage, and ultimately die.

In parallel with these MedSAF scenario activities, the HPS may be remotely connected to MedSAF. Commands over the modem connection can initiate communications and initialize the HPS in a given treatment scenario. Medical practitioners can provide treatment to the HPS in the form of intubation, anesthesia, and injection of drugs. In conjunction with this, real-time vital sign data from the HPS is transmitted over the modem connection and displayed to the MedSAF operator via the Data Grapher.

6. Future Work

There are many applications of our medical simulation capabilities. While our prototype is directly applicable to the areas of combat medical readiness for all services in the Department of Defense, including combat medical training and supporting a materiel development environment for evaluation of different medical doctrines, other applications are also possible. For example, training for mass casualty triage and treatment could greatly benefit from validated simulations that can generate realistic casualties based on realistic scenarios. Other examples of potential future applications are described below.

6.1 2-Way HPS Linkage

A complete two-way linkage between MedSAF and the HPS is the next logical developmental step. Under this linkage, particular casualties generated in MedSAF will be downloaded to the HPS over the modem communications link. Sufficient medical state must be forwarded to the HPS to load it with a useful medical training scenario that correlates between the MedSAF representation of the casualty and the HPS. Also under this complete linkage, the MedSAF DataGrapher should be integrated into MedSAF so that the operator can direct the display of MedSAF casualty vital-signs as well as HPS communicated vital-signs. This facility would aid in performing validation scenarios which would be used to correlate the MedSAF and HPS medical models. In addition, a continuous linkage concept could be provided, which would allow one MedSAF scenario to generate multiple casualties which are transported to various treatment locations on the synthetic battlefield. Certain casualties could be downloaded to the HPS on demand by a scenario controller to provide medical training to trainees co-located with the HPS. As part of triage training, multiple HPS's could be used to represent multiple casualties, with medical personnel deciding whether to treat a high priority patient immediately or to move on to the next patient. This evolves naturally into a mass casualty training and experimentation laboratory.

6.2 Enhanced Medical Scenarios

The MedSIM system provides opportunities for enhanced medical scenarios. MedSAF supports environmental modeling that includes battlefield smoke and may ultimately include chemical and other environmental agents (Schaffer 1994). Treating casualties inflicted with injuries due to smoke or chemical warfare is already possible using the HPS; slight extensions are needed to MedSAF to play NBC and smoke.

Another possible direction for new medical scenarios include simulation of mass casualties due to man-made accident (such as a passenger plane crash) or natural disasters (such as earthquake). Appropriate statistical distributions of casualty type could be modeled, and MedSAF capabilities of dispatching care and evacuation and HPS capabilities of simulating triage and accepting clinical treatment can be used to train coordinators, medics, and clinicians in a mass-casualty scenario.

6.3 Other Linkages

As described in our previous report, a linkage to TCIMS' Mobile Medical Monitor (MMM) is possible. The MMM could be attached to the HPS to

monitor real-time vital-sign data. The combination would be used by selected medical personnel to assess and subsequently treat the HPS. This scenario would provide a proof-of-concept of several integrated capabilities: (1) improved training, with real time feedback and response; (2) algorithm validation; (3) integration of medical teams into synthetic combat exercises; and (4) real time testing and evaluation of medical readiness using the HPS to supply realistic test data.

7. Conclusions

We have presented our vision for using simulation linkages to improve medical readiness, and discussed the prototypes used to prove the viability of the concept. We have described how these capabilities can be implemented within the DIS and CGF paradigm. The success of our work so far has convinced us that this approach is sound. Although many challenges still face us in the development of synthetic medical environments, we believe this work has helped to establish a clear roadmap to a mass casualty training system using integration approaches developed for the combat community.

8. References

- Aronson, J. (1996). "The STOW97 System Architecture and Implementation Design", *Proceedings of the 14th Workshop on Standards for the Interoperability of Distributed Simulations*, Orlando, FL: Institute for Simulation & Training. pp. 447-454.
- Calder, R. B., Smith, J. E., Courtemanche, A. J., Mar, J. M. F., and Ceranowicz, A. Z. (1993). "ModSAF Behavior Simulation and Control", *Proceedings of the Third Conference on Computer Generated Forces and Behavioral Representation*, Orlando, FL: Institute for Simulation & Training. pp. 347-356.
- Courtemanche, A. J., Bimson, K., van Oostrom, J., and Lampotang, S. (1996). "A DIS-Compatible Medical Simulation Environment for the Battlefield", *Proceedings of the MEDTEC Conference*, Orlando, FL.
- Courtemanche, A. J., and Ceranowicz, A. (1995). "ModSAF Development Status", *Proceedings of the Fifth Conference on Computer Generated Forces and Behavioral Representation*, Orlando, FL: Institute for Simulation & Training. pp. 3-13.
- Courtemanche, A. J., and Monday, P. (1994). "The Incorporation of Validated Combat Models into ModSAF", *Proceedings of the Fourth Conference on Computer Generated Forces and Behavioral Representation*, Orlando, FL: Institute for Simulation & Training. pp. 129-140.

- Lampotang S, Good ML, van Meurs WL, Carovano RG, Azukas J, Rueger EM, Gravenstein JS (1995). "The University of Florida/Loral human patient simulator, abstracted" *J Anesthesia* 9:SS1-5.
- Schaffer, R. (1994). "Environmental Extensions to ModSAF", *Proceedings of the Fourth Conference on Computer Generated Forces and Behavioral Representation*, Orlando, FL: Institute for Simulation & Training. pp. 17-23.
- van Meurs WL, Beneken JEW, Good ML, Lampotang S, Carovano RG Jr., Gravenstein JS (1993). "Physiologic model for an anesthesia simulator, abstracted" *Anesthesiology* 79:A1114.

9. Biographies

Anthony J. Courtemanche has over eight years of experience in software engineering for SIMNET and DIS programs using SAF applications. His software accomplishments include SAF architecture, weapons systems simulation, targeting behaviors, network simulation protocols, and user interfaces. Mr. Courtemanche was one of the principal contributors to the ModSAF architecture, including the Persistent Object Protocol. At Loral ADS, he was the Project Engineer for ADST ModSAF System Development Delivery Order. Now at SAIC, Mr. Courtemanche is a Senior Scientist in the Advanced Distributed Simulation Research Team, and he supports ADS IRAD projects and new business development in the areas of Semi-Automated Forces and medical simulation, as well as supporting the ADST-II program as the Senior Systems Engineer of the ModSAF Engineering Team. Mr. Courtemanche received his Master of Science degree in Electrical Engineering from M.I.T. in 1987.

Kent Bimson, Ph.D., is Chief Scientist at Science Applications International Corporation (SAIC) Orlando. Dr. Bimson is in charge of coordinating Group research and for business development in support of the group's research efforts, including medical simulation. Dr. Bimson formerly served Chief Scientist, Artificial Intelligence for the Semi-Automated Forces (SAF) component of the U.S. Army's Close Combat Tactical Trainer (CCTT) program. Before joining SAIC, Dr. Bimson served as a Research Scientist at Lockheed Software Technology Center in Austin, Texas from 1985 through 1991, where his research focused on knowledge based technologies. Dr. Bimson served as Associate Professor of Computer Science at California State University, Sacramento (CSUS) from 1981 through 1985, where he taught courses in artificial intelligence and natural language processing. He holds a Ph.D. in Linguistics from UCLA (1978) and a Masters in Computer Science from CSUS (1983).

A Model of Large-Scale Citizen Evacuation for Emergency Management Simulation

Ross C. Creech and Mikel D. Petty
Institute for Simulation and Training
3280 Progress Drive
Orlando FL 32826-0544 USA
mpetty@ist.ucf.edu

1. Abstract

The response to large-scale emergencies can involve large numbers of personnel, vehicles, and other resources. Such responses are controlled and managed during and after an emergency by emergency managers. The Plowshares project enhanced Janus, a military constructive simulation, to produce TERRA, a simulation that can be used to train emergency managers in a command post exercise format. An important activity during a large-scale emergency is the evacuation of the civilian populace in the affected area. This paper describes the design and implementation status of the evacuation model designed for Plowshares.

2. Introduction

This section provides background on the Plowshares project, which was the context for the evacuation model that is the subject of this paper, and briefly surveys some previous research in evacuation modeling.

2.1 The Plowshares project

Large-scale emergencies, such as earthquakes and hurricanes, require massive responses, involving large numbers of personnel, vehicles, and other resources. Emergency managers are charged with managing and allocating resources and coordinating the many actions taken in response to an emergency. The Plowshares project applied military constructive simulation technology to produce a simulation intended to train emergency managers. In particular, the U. S. Army's Janus entity-level constructive simulation model (Titan 1993) was enhanced with emergency management features. The resulting simulation, called TERRA, simulates an emergency and the actions taken in response to it, allowing emergency managers to practice their skills. TERRA is used in a command post exercise mode, where the command hierarchy and communications channels of emergency managers remain unchanged, except at the lowest level, where actual disaster events (such as fires) and response units (such as fire trucks) are replaced with the computer

simulation. The initial version of TERRA simulated the effects of hurricanes, fires, tornadoes, chemical spills, and other hazards, and response actions such as fire fighting and road clearing.

More information on the Plowshares project can be found in any of the following:

1. Project overview (Petty 1996) (Petty 1995b)
2. CGF capabilities needed for emergency management simulation (Petty 1995a)
3. Emergency management training using simulation (Slepow 1995)
4. Mathematical models of disaster events (Wood 1995)

For many types of emergencies, the large-scale evacuation of citizens is a major factor. Thousands of people fleeing a hurricane completely occupy the transportation network in the affected area. Controlling and facilitating that evacuation requires communication and coordination among all emergency response agencies. Effective training of emergency managers with a simulation requires that the large-scale evacuation of citizens be modeled. This paper presents the evacuation model designed for TERRA and reports the status of its implementation.

2.2 Evacuation modeling

A number of models have been proposed and developed to cover different types of evacuations, employing a variety of different modeling methods. References to some of that work are listed below:

1. Common characteristics of evacuation models (Banz 1991)
2. Optimal egress modeling as a state dependent finite closed queuing network (Bakuli 1991)
3. Building evacuation, based on network flow (Choi 1991)
4. Regional evacuation on roads (Newsom 1991)
5. Evacuation decision support (Kisko 1991)
6. Evacuation around a nuclear power station, based on network flow on roads (Hobeika 1991)
7. Survey of evacuation models and methods (Lovas 1993)

8. Mine evacuation during a mine fire, based on network flow (Unger 1993)
9. Evacuation of a geographical area in advance of a hurricane, based on network flow on roads (Tufceki 1993)
10. Improved calculation of travel speed along arcs in network flow evacuation models (Bakuli 1993)
11. Evacuation movement of human flow modeled as particle systems (Bouvier 1995)

3. Evacuation model design

This section presents the evacuation model design.

3.1 Design overview

The Plowshares evacuation model does not represent individual persons and their specific locations. Rather, it represents the geographic area to be evacuated as a two dimensional array of square cells, each 100x100 meters. An attribute of each cell is the number of persons in that cell. During execution, persons flow from cell to cell at discrete time intervals according to the constraints of the underlying terrain, moving away from hazards (such as fires or hazardous chemicals) and towards safety (such as shelters). This familiar finite element method for modeling continuous flows has been widely used in applications as diverse as heat flow (Jacoby 1980) and tornado winds (Davies-Jones 1995).

3.2 Assumptions

The evacuation model design makes certain assumptions, which are listed here. Essentially, each assumption is a reflection of what information the underlying emergency management simulation must provide to the evacuation model.

1. *Presence of hazards.* The emergency management simulation must set hazard flag(s) in affected cells to note the presence of fire, obstacles, and hazardous materials, and change those flags over time as the extent and location of the hazards change.

2. *Presence of shelters.* The emergency management simulation must note the presence and capacity of a shelter in a given cell and its capacity.

3. *Initial cell population.* The emergency management simulation must initialize each cell with an initial population and maximum capacity.

4. *Presence of roads.* The emergency management simulation must initialize each cell with or without the presence of roads.

5. *Casualty parameters.* The probabilities of each hazard causing casualties must be given as parameters.

6. *Discrete cell updates.* The state of all grid cells' attributes are assumed constant during any given evacuation time interval. Although an attribute could change value many times during the time step; only its final value is used for the evacuation model. This assumption applies only if the evacuation model uses a time step that is different from the underlying emergency management simulation.

3.3 Terrain grid cell attributes

The citizen evacuation model relies heavily on the database design of the Plowshares TERRA software. In the TERRA, citizens are made an attribute of the terrain. The terrain map is divided into cells of 100 square meters. The size of the terrain map determines the number of cells; a 60 square kilometer map has a 600x600 grid of cells.

Each cell is one of three feature types: generic urban, vegetation, or generic areas. Each area feature type has seven classes. Each cell's value of type and class determine an initial population for that cell. In addition to the feature type/class and initial population, several other attributes will be used in the citizen evacuation model. They are listed in Table 1; the table also shows how each attribute is initially set and whether the attribute is dynamic or static.

3.4 Preprocessing

Preprocessing refers to all processing that takes place before the evacuation model begins execution. Note that the emergency management simulation execution begins prior to the start of the evacuation model, during which time the initial effects of the disaster (e.g. hurricane) are calculated. The preprocessing steps are as follows:

1. Initialize feature type and class.
2. Initialize presence of buildings, fences, rivers, obstacles, roads, and shelters.
3. Initialize population and maximum capacity.
4. Hurricane enters the area.
5. Update presence of hazards (fire, obstacles, hazardous materials) and shelters.
6. Assess casualties and update population.
7. Hurricane leaves the area (the hurricane can have moved far enough away to begin the exercise; however, it can still cause damage).
8. Initialize population for evacuation model with each cell's current population
9. Initialize basic flow rate of citizens.

3.5 Feature type/class and roads

Each cell's feature type and the presence of roads are established in the terrain database. These attributes are predefined before the simulation ever begins and never change. Global variables identify a cell's feature type and the presence of primary or secondary roads. Bit masks are used to determine the presence of roads and the associated feature type.

3.6 Number of citizens

The initial number of citizens per cell is also preprocessed; however, each cell's initial value to be used by the citizen evacuation model is calculated after the hurricane leaves the area and the training exercise begins. As mentioned previously, each cell has an initial population based on its feature type and class. As the hurricane moves through the population of affected cells will decrease. Once the hurricane leaves the area, the resulting population of each cell becomes the initial value to be used by the evacuation model. Casualties inflicted by the hurricane are tracked separately from those resulting from the evacuation.

3.7 Hazards

Hazards are created by the hurricane and the state of any hazard can change over time. Models will set/reset flag(s) to note the presence of a hazard in a given cell; bit masks are used to determine each hazard's presence.

3.8 Flow rate

Each cell will have an associated flow rate of citizens across that cell. This flow rate determines how many citizens can move across the cell in one time step. A cell's basic flow rate is a function of its feature type; the basic flow rate is dynamically modified by cell attributes to calculate at each time step an adjusted flow. Table 2 shows the basic flow rate for cells based on their feature type. Table 3 shows the general effect of each cell attribute on the cell's adjusted flow rate. The adjusted flow rate will take into account the presence of hazards, roads, and police, and will be the rate that is used to move citizens. The number of citizens to move is the adjusted flow rate times the number of time steps since the last cell update. However, the number of citizens to evacuate can not exceed the cell's current population and can not exceed the maximum population of the cell being evacuated to.

c_1 = current cell's population (citizens)
 c_2 = cell to evacuate to's population (citizens)
 m_2 = cell to evacuate to's max population (citizens)
 n = number of citizens to evacuate (citizens)
 r_b = cell's basic flow rate (citizens per minute)
 r_a = cell's adjusted flow rate (citizens per minute)
 Δt = time since last cell update (minutes)

$$n = \text{MIN}[r_a \cdot \Delta t, c_1, m_2 - c_2]$$

3.9 Population capacity and shelters

The maximum population capacity attribute limits the number of persons that may be present in a cell. A cell's maximum population will be a function of the cell's feature type, class, and the presence of a shelter. The cell's feature type and class will determine a maximum population assuming no shelter exists in the cell. The presence of a shelter adds to the maximum population capacity. It should be noted that since shelters can be destroyed at any time during the exercise, the cell's maximum population can change. If the cell population exceeds the cell's maximum shelter population, it is assumed that the shelter is full and the others are unsheltered.

3.10 Attraction index

Each cell will have an associated attraction index which determines its likelihood to attract citizens. Citizens move towards cells with larger attraction indices. Population to or from any area can be caused by a variety of factors. (Banz 1991) lists some of those and notes that some cause movement away from a location (e.g. hazards) and others cause movement towards a location (e.g. safety). This general idea is extended to the concept of attractors and repellers. As previously mentioned, each cell has an associated attraction index; and citizens move toward cells with larger attraction indices. Each cell's attraction index is a function of nearby repellers and attractors. Citizens move away from repellers and towards attractors; see Table 5.

Table 6 shows the effect of repellers on a cell's attraction index, based on the current cell's distance from the repeller source. The numbers 0 - 5 indicate the number of cells (range) away from the repeller source. A range of 0 indicates the effect of a repeller located in that cell; a range of 5 indicates the effect of a repeller located 5 cells away. Table 7 shows the effect of attractors on a cell's attraction index, based on the current cell's distance from the attractor source.

Grid Cell Attribute	Initialization	Dynamic?
Feature Type / Class - Generic Urban - Vegetation - Generic Areas	Preprocess	No
Road - Primary - Secondary	Preprocess	No
Hazard(s) - Fire - Obstacle - Hazardous materials	By hurricane	Yes
Number of citizens	Preprocess	Yes
Maximum population capacity	Preprocess	Yes
Flow rate	Preprocess	Yes
Shelter capacity	Preprocess	Yes
Attraction index	None	Yes

Table 1. Grid cell attributes.

Feature Type	Basic Flow Rate
Generic Urban	1000
Vegetation	2000
Generic Areas	1500

Table 2. Basic cell flow rate.

Attribute	Effect	Comment
Feature Type	Positive	Vegetation (relative to generic)
	Negative	Urban (relative to generic)
Number of Citizens	Negative	Crowded cells slow movement
Presence of Hazards	Negative	
Presence of Roads	Positive	
Presence of Police	Positive	Police assigned to traffic control
	Negative	Police cordon around hazard

Table 3. Effect of cell attributes on flow rate.

Attribute	Presence	Adjustment Factor
Fire	No	1.0
	Yes	0.5
Obstacle	No	1.0
	Yes	0.7
HAZMAT	No	1.0
	Yes	0.5
Road	None	1.0
	Primary	1.8
	Secondary	1.6
Police to increase flow	0	1.0
	1 - 2	1.5
	3 - 5	2.0
	>= 6	2.2
Police to block flow	0	1.0
	1 - 2	.5
	3 - 5	.3
	>= 6	.2
Citizens	0 - 1/4 Full	1.0
	1/4 - 1/2 Full	.8
	1/2 - 3/4 Full	.5
	3/4 - Full	.3

Table 4. Flow rate adjustment factors (notional).

Repellers	Attractors
Hazards	Shelters
Crowded Cells	Uncrowded Cells
	Cells with Roads

Table 5. Repellers and attractors.

Repellers	Range (in cells)					
	0	1	2	3	4	5
Fire	-7	-4	-1	0	0	0
Rubble	-6	-3	0	0	0	0
Obstacle	-6	-3	0	0	0	0
HAZMAT	-6	-4	-2	-1	0	0
Crowded cell	-6	-3	0	0	0	0

Table 6. Repellers' effect on a cell attraction index.

Attractor	Range (in cells)					
	0	1	2	3	4	5
Shelter	2	1	0	0	0	0
Primary road	2	1	0	0	0	0
Secondary road	2	1	0	0	0	0
Uncrowded cell	2	1	0	0	0	0

Table 7. Attractors' effect on a cell attraction index.

Similar to Table 6, the numbers 0 - 5 indicate the number of cells (range) away from the attractor source. The parameter values chosen in Tables 4, 6, and 7 are notional. The actual values of these parameters will be determined with the aid of subject matter experts data collected from actual disaster evacuations.

A cell's attraction index is the cumulative effect of all attractors and repellers within range. The calculation is:

$I = \text{Attraction Index}$

$na = \text{Number of Attractors Within Range}$

$np = \text{Number of Repellers Within Range}$

$a_i = \text{Value for Attractor } i$

$p_j = \text{Value of Repellor } j$

$$I = \sum_{i=1}^{na} a_i + \sum_{j=1}^{np} p_j$$

3.11 Algorithm overview

During each time step of TERRA:

- (1) As hazards are created/removed, update cell.
- (2) For each cell, assess casualties.
 - (2.1) For each hazard present in the cell,

$$\text{casualties} = \text{number of citizens} * \text{probability of casualty}$$

$$\text{number of citizens} = \text{number of citizens} - \text{casualties}$$

During each time step of the evacuation model:

- (1) For each cell, calculate its adjusted flow rate.
- (2) For each cell, calculate its attraction index.
- (3) For each cell, move citizens to adjacent cells (if possible and necessary).

Rules for movement:

 - (3.1) Citizens will move to the cell with the greatest attraction index.
 - (3.2) If all cells have the same attraction index, the citizens will not move.
 - (3.3) Citizens can not move to a cell whose maximum population capacity would be exceeded.
 - (3.4) The number of citizens to move is the minimum of the cell's current population, the maximum number of citizens that could move via the calculation of n, and the available space in the cell to move to.
 - (3.5) If 2 or more cells have the same largest

attraction index,

If one of the cells under consideration is the citizens' current location, then the citizens will not move, else the citizens will move to the least crowded cell first, the second least crowded cell next, and so forth until all citizens are moved or all cells under consideration are full. Any remaining citizens will not move.

3.12 Training characteristics

With this evacuation model, the emergency managers are challenged to allocate police to control traffic and optimize the evacuation flow. They must also reduce casualties during the evacuation by eliminating the hazards (e.g. using fire trucks to extinguish fires) and using police to direct evacuation flow away from and around hazards. The emergency managers can be measured quantifiably based on the number of casualties versus total population and the number of sheltered versus unsheltered citizens.

4. Evacuation model implementation

This section details the initial implementation of the evacuation model into the Flowshares TERRA software. The initial implementation was completed under tight time constraints and includes only limited functionality; however, it serves as a basis for further development. The design and implementation of the initial evacuation model are discussed, as well as suggested improvements for subsequent development iterations.

4.1 Chemical cloud hazard

A single chemical cloud is the only hazard used in the initial evacuation model implementation. The chemical model incorporated within the TERRA software, which was not modified from Janus, is used to update the chemical cloud's location, radius, and toxicity. These parameters are dynamic and can change with each chemical model update.

4.2 Flow rate calculation

The evacuation model uses a cellular approach, dividing the terrain into a grid of cells, each with an associated feature type occupying 100 square meters. The terrain editor is used to associate different terrain feature types with the population density, measured in citizens per square kilometer, basic flow rate of citizens across the cell, measured in citizens per minute.

Terrain feature types include urban, vegetation, and generic areas. As noted in Figure 1, all cells that lie within 0.5 kilometers of the chemical cloud boundary are evacuated. All other cells are assumed to be within a safe distance and evacuation is unnecessary. The cell center is used for all calculations. The distance of 0.5 kilometers is a notional number that can be revised as deemed necessary. For those cells that lie within the radius of the cloud, the flow rate of citizens increases by 25% in order to allow for increased movement due to citizen panic. This increase is also notional and can be revised as necessary. Finally, the number of citizens to evacuate can not exceed the cell's current population.

Determining if a cell lies within the evacuation area is done based on simple Euclidean distance. We begin with the following variable definitions:

- (x_1, y_1) = coordinate of cloud center
- (x_2, y_2) = coordinate of cell center
- r = radius of cloud
- d_1 = distance from cell center to cloud center (km)
- d_2 = distance from cell center to cloud boundary (km)
- c = cell's current population (citizens)
- n = number of citizens of evacuate (citizens)
- r_b = cell's basic flow rate (citizens per minute)
- r_a = cell's adjusted flow rate (citizens per minute)
- Δt = time since last cell update (minutes)

Using the diagram in Figure 2 as a reference, d_1 and d_2 are calculated using Equations (1) and (2), respectively.

$$d_1 = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} \quad (1)$$

$$d_2 = d_1 - r \quad (2)$$

In summary, the pseudocode for calculating the adjusted flow rate and number of citizens to evacuate for each cell is:

```

if( $d_2 < 0.5$ )
{
  if( $d_2 < 0$ )  $r_a = 1.25 \cdot r_b$ 
  else  $r_a = r_b$ 
   $n = \text{MIN}[r_a \cdot \Delta t, c]$ 
}

```

4.3 Evacuation movement direction

Each cell is examined in relation to the cloud center and the wind direction. For analytic purposes, the cell center and chemical cloud center are used for all calculations. As shown in Figure 3, each cell and the wind direction can lie in 1 of 8 regions. Each region occupies a portion of the terrain spanning an arc of 45° from the chemical cloud center with the size of each region dependent on the location of the chemical cloud within the terrain boundary. The direction in which the citizens evacuate is determined by the cell's region and the chemical cloud's region. Evacuation movement of citizens is modeled as movement from cell to cell. Citizens will evacuate to one of the surrounding cells to the north, northeast, east, southeast, south, southwest, west, or northwest. In certain circumstances (i.e. for a cell in the corner), not all eight directions are available for evacuation. See Table 8.

The chemical cloud's region is determined by the direction of the wind. The direction is merely an angle measured from the +x axis, referred to as q_1 in the diagram in Figure 2. For example, a wind direction of 225° degrees places the chemical cloud in region 6. The cell's region is also determined with a direction, q_2 ; but this direction is dynamic and varies from cell to cell. Angle q_2 is the angle measured from the cloud center to the cell center (measured from the +x axis). Using the diagram in Figure 2 as a reference, one can calculate q_2 using Equation (3). For instance, a cloud center to cell center direction of 45° places the cell in region 2. Assuming this cell lies within the evacuation area and the chemical cloud is in region 6, citizens will move to the cell to the northeast.

$$\Theta_2 = \text{TAN}^{-1}\left(\frac{y_2 - y_1}{x_2 - x_1}\right) \quad (3)$$

If the chemical cloud's region and the cell's region are identical, the evacuation direction is determined by noting which direction (angle) is larger. See Table 8. It should be noted, however, that Table 8 provides only one evacuation direction. If for some reason, this direction is unavailable, the citizens can not evacuate. In subsequent development iterations, the table should be extended to allow for alternative directions if the first choice is unavailable.

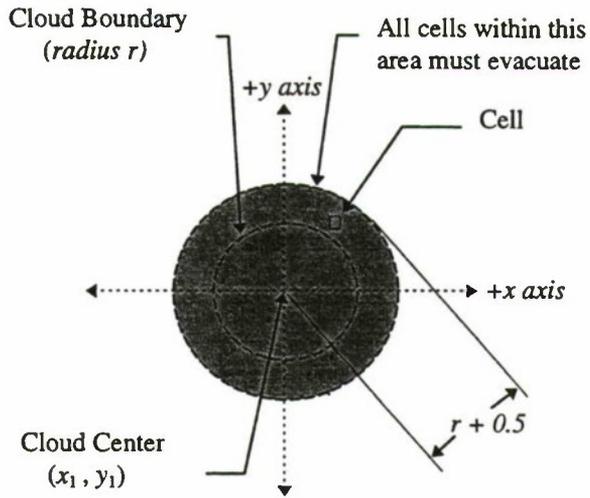


Figure 1. Evacuation area.

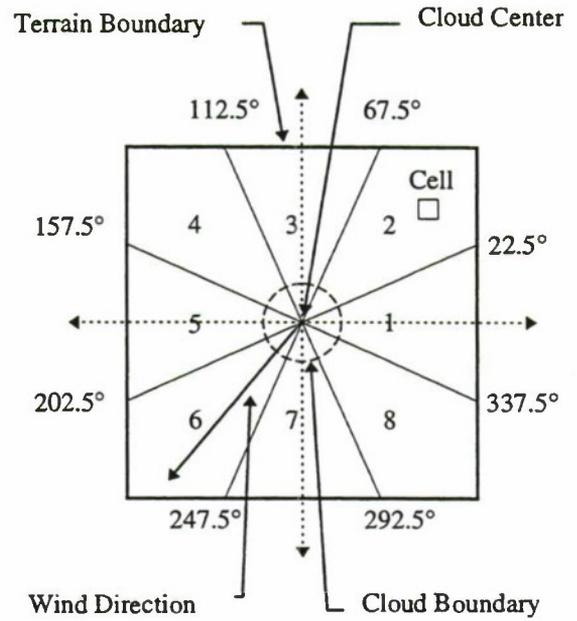


Figure 3. Evacuation direction regions.

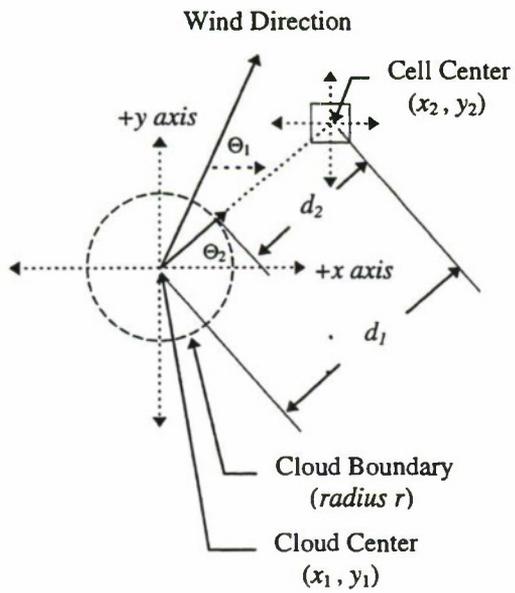


Figure 2. Evacuation distance calculation.

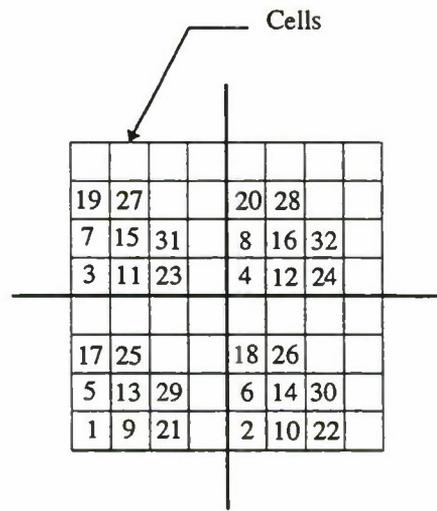


Figure 4. Cell evaluation sequence.

Chemical cloud region (q1)		Chemical cloud center to cell center region (q2)							
		1	2	3	4	5	6	7	8
1	$q_2 \geq q_1 \rightarrow N$ $q_2 < q_1 \rightarrow S$	N	N	NW	NW	W	SW	S	S
2	SE $q_2 \geq q_1 \rightarrow NW$ $q_2 < q_1 \rightarrow SE$	$q_2 \geq q_1 \rightarrow NW$ $q_2 < q_1 \rightarrow SE$	NW	NW	W	SW	SW	SE	SE
3	E	E	$q_2 \geq q_1 \rightarrow W$ $q_2 < q_1 \rightarrow E$	W	W	W	SW	S	SE
4	E	NE	NE	$q_2 \geq q_1 \rightarrow SW$ $q_2 < q_1 \rightarrow NE$	SW	SW	SW	S	SE
5	E	NE	N	N	$q_2 \geq q_1 \rightarrow S$ $q_2 < q_1 \rightarrow N$	S	S	S	SE
6	N	NE	E	NW	NW	$q_2 \geq q_1 \rightarrow SE$ $q_2 < q_1 \rightarrow NW$	SE	SE	SE
7	E	NE	N	NW	W	W	W	$q_2 \geq q_1 \rightarrow E$ $q_2 < q_1 \rightarrow W$	E
8	NE	NE	N	NW	W	W	SW	SW	$q_2 \geq q_1 \rightarrow NE$ $q_2 < q_1 \rightarrow SW$

Table 8. Evacuation direction.

4.4 Integrating the evacuation model into TERRA

The TERRA simulation software is organized around a number of event queues. Event queues include events for hurricane, fire, tornado, vehicle movement, etc. Each event queue contains a series of events to occur in the future, each with an associated time stamp, that determines event.

The evacuation model is implemented as another event queue, very similar to the design of hurricane disaster model. The TERRA software is easily enhanced to include an additional event queue within the main simulation driver. The manner in which the initial evacuation model implementation examines each cell is identical to that of the hurricane model. The hurricane model must assess damage for all cells. Due to the large number of cells, the model can not update all cells at each model update. Therefore, the hurricane model uses an update interval which specifies the time it takes to update all cells on the terrain.

Rather than evaluate the cells sequentially from bottom to top, the terrain is divided into four quadrants. One cell from each quadrant is evaluated for each hurricane damage update iteration. The evaluated cells form a grid across the terrain and evenly distribute the damage evaluation for each iteration. The starting point determines which cell in each quadrant is evaluated. The starting point moves up and over alternatively, generating a wave-like evaluation from lower left to upper right in each quadrant. Figure 4 details the order in which the first 32 cells are evaluated.

It should be noted, however, that in using this approach for the initial evacuation model implementation, all cells are analyzed, regardless of their proximity to the chemical cloud. As previously mentioned, only those cells with 0.5 kilometers of the cloud boundary require evacuation, and much time is wasted checking those cells outside of this area.

In subsequent development iterations, the same interleaved cellular examination approach can be used; but rather than examine all cells, only those cells within the bounding box surrounding the chemical cloud need to be checked. Given the location and radius of the chemical cloud, one can easily calculate the corner cells that bound the evacuation area.

4.5 Future improvements

Several unimplemented features of the design and possible improvements remain with the initial evacuation model implementation. Some of those are listed here:

1. *Cell capacity.* The current model allows for an infinite cell capacity. The software should be enhanced to give each cell a maximum capacity based on its associated terrain feature type and the presence of additional buildings and/or shelters.

2. *Presence of hazards.* The current model only recognizes one hazard source, a single chemical cloud. The model should be enhanced to recognize multiple hazards.

3. *Presence of attractors.* The current model does not recognize any attractors. The model should be enhanced to recognize multiple attractors to attract citizens. Attractors include the presence of shelters and/or roads.

4. *Presence of emergency response units.* The current model does not recognize the presence of emergency response units and police. The model should be affected by the presence of these units.

5. *Attraction index calculation.* The current model does not incorporate the attraction index calculation. After being able to recognize multiple hazards, attractors and the presence of emergency response vehicles, this calculation can be incorporated.

6. *Cell evaluation algorithm.* Rather than implement the algorithm detailed in section 3.4, a more sophisticated algorithm should be developed that uses multiple hazard locations to determine which cell to evacuate next. In addition, this algorithm should only evaluate those cells that require evacuation.

7. *Discrete events.* Ideally, evacuation of multiple cells would occur simultaneously; however, in a discrete simulation, such events must occur sequentially. Discrete, sequential calculation can cause undesired artificial delays to movement when cell populations are at or near the cell's capacity.

5. Conclusions

Large scale civilian evacuation is an important part of emergency managers' responsibilities, and must be included in an emergency management simulation. A relatively simple and elegant model can provide a usefully realistic representation of evacuation.

6. Acknowledgments

This work was partially supported by the U. S. Army Simulation, Training, and Instrumentation Command as part of the Plowshares project, contract N61339-96-K-0003, under the supervision of STRICOM Project Director Jean H. Burmester. That support is gratefully acknowledged.

7. References

- Bakuli, D. L. and Smith, J. M. (1991). "Optimal Routing and Resource Allocation within State Dependent Evacuation Networks", *Proceedings of the SCS Multiconference on Simulation in Emergency Management and Engineering and Simulation in Health Care*, Anaheim CA, January 23-25 1991, pp. 23-30.
- Bakuli, D. L. and Smith, I. M. (1993). "Optimal Routing in State Dependent Evacuation Networks", *Proceedings of the 1993 International Emergency Management and Engineering Conference*, Arlington VA, March 29 - April 1 1993, pp. 87-90.
- Banz, George (1991). "Toward a Generic Evacuation Simulation Technique", *Proceedings of the SCS Multiconference on Simulation in Emergency Management and Engineering and Simulation in Health Care*, Anaheim CA, January 23-25 1991, pp. 39-41.
- Bouvier, E. and Cohen, E. (1995). "Simulation of Human Flow with Particle Systems", *Proceedings of the 1995 Simulation MultiConference*, Phoenix AZ, April 9-13 1995, pp. 349-354.
- Choi, W. (1991). "A Simulation Model for Emergency Building Evacuation", *Proceedings of the SCS Multiconference on Simulation in Emergency Management and Engineering and Simulation in Health Care*, Anaheim CA, January 23-25 1991, pp. 31-38.
- Davies-Jones, R. (1995). "Tornadoes", *Scientific American*, Vol. 273, No. 2, August 1995, pp. 48-57.
- Hobeika, A. G. and Kim, S. (1991). "Emergency Evacuation Around Nuclear Power Stations", *Proceedings of the SCS Multiconference on Simulation in Emergency Management and Engineering and Simulation in Health Care*, Anaheim CA, January 23-25 1991, pp. 54-61.
- Jacoby, S. L. S. and Kowalik, J. S. (1980). *Mathematical Modeling with Computers*, Prentice-Hall, Englewood Cliffs NJ, 1980.
- Kisko, T. and Tufekci, S. (1991). "Design of a Regional Evacuation Decision Support System: Integrating Simulation and Optimization", *Proceedings of the SCS Multiconference on Simulation in Emergency Management and Engineering and Simulation in Health Care*, Anaheim CA, January 23-25 1991, pp. 48-53.
- Lovas, G. G., Wilklund, J., and Drager, K. H. (1993). "Evacuation Models and Objectives", *Proceedings of the 1993 International Emergency Management and Engineering Conference*, Arlington VA, March 29 - April 1 1993, pp. 91-97.
- Newsome, D. E. and Beriwal, M. (1991). "Regional Evacuation Planning Using Computer Simulation: Promise and Pitfalls", *Proceedings of the SCS Multiconference on Simulation in Emergency Management and Engineering and Simulation in Health Care*, Anaheim CA, January 23-25 1991, pp. 42-47.
- Petty, M. D., Slepow, M. P., and West, P. D. (1995a). "CGF Opportunities in Plowshares", *Proceedings of the Fifth Conference on Computer Generated Forces and Behavioral Representation*, Orlando FL, May 9-11 1995, pp. 337-344.
- Petty, M. D. and Slepow, M. P. (1995b). "Plowshares: Emergency Management Training with a Military Constructive Simulation", *Proceedings of the 17th Interservice/Industry Training Systems and Education Conference*, Albuquerque NM, November 13-16 1995.
- Petty, M. D. and Slepow, M. P. (1996). "Plowshares: An Emergency Management Training Simulation", *Simulation*, Vol. 66, No. 6, June 1996.
- Slepow, M. P. and Kincaid, J. P. (1995). "Plowshares: Effective Training Using an Emergency Management Simulation", *Proceedings of the 1995 Southeastern Simulation Conference*, Orlando FL, October 22-24 1995, pp. 141-149.
- Titan, Inc. *The Janus 3.X/UNIX Model User's Manual*, W800XR0-3125-0052, TRADOC Analysis Center.
- Tufekci, S., Sandesh, J. J., Albusairi, A. (1993). "Importance of REMS in the Aftermath of Hurricane Andrew", *Proceedings of the 1993 International Emergency Management and Engineering Conference*, Arlington VA, March 29 - April 1 1993, pp. 81-86.
- Unger, R. L., Glowacki, A. F., and Stein R. R. (1993). "An Evacuation Simulation for Underground Mining", *Proceedings of the 1993 International Emergency Management and Engineering Conference*, Arlington VA, March 29 - April 1 1993, pp. 75-80.
- Wood, D. D., Farr, J. V., Horsley, M., and Petty, M. D. (1995). "Plowshares: Hurricane, Tornado, and Fire Modeling in TERRA", *Proceedings of the 1995 Southeastern Simulation Conference*, Orlando FL, October 22-24 1995, pp. 159-166.

8. Authors' biographies

Ross C. Creech is a Research Assistant at the Institute for Simulation and Training, working on the High Level Architecture BDS-D project. Previously he worked on the software engineering team for the Plowshares project. Mr. Creech recently received a M.S. at the University of Central Florida in Computer Engineering, specializing in Software Engineering. He also holds a B.S. in Industrial and Systems Engineering from the University of Florida and has professional experience in multi-chip microelectronics design and manufacturing.

Mikel D. Petty is a Program Manager and Senior Research Computer Scientist at the Institute for Simulation and Training. He is currently leading IST's HLA BDS-D project; previously he managed IST's Emergency Management and Computer Generated Forces research. Mr. Petty received a B.S. in Computer Science from the California State University Sacramento and a M.S. in Computer Science from the University of Central Florida, and is a Ph.D. student in Computer Science at UCF. His research interests are in simulation and computational geometry.

Application of Computer Generated Force Technology to Interagency Drug Interdiction

John Miller and Greg Jackson
BMH Associates, Inc
5425 Robin Hood Road, Suite 201
Norfolk, VA 23513-2441
miller@bmh.com jackson@bmh.com

Will Miller
Joint Interagency Task Force East
Key West, FL

1. Abstract

Over several years and administrations, the U.S. government's drug interdiction strategy has evolved to an approach that emphasizes the selective, intelligence-cued, and carefully planned employment of a constrained number of interdiction assets in the transit zones (e.g. the Caribbean air and sea routes) leading to the continental United States. Given their limited resources, the three Joint Interagency Task Forces (JIATF) East, South, and West, established in 1994 would greatly benefit from the application of Computer Generated Force (CGF) technology to the training and operational tasks implicit in their interdiction mission. In the training arena, the JIATFs are responsible for integrating law enforcement and military personnel of varied expertise and experience into cohesive command center teams capable of smooth, effective action to counter detected air and maritime trafficking events. Operationally, the JIATFs continually face cost-benefit decisions in determining the optimal force laydown, near term and long range, to counter drug trafficking trends. Moreover, with reliable pre-event intelligence, the JIATFs conduct detailed planning and gaming to ensure that limited assets are most effectively arrayed against anticipated specific events. Adapting the CGF capability to replicate air and maritime interdiction operations for training, mission rehearsal and after action analysis purposes could pay a substantial dividend in this critical national and international security issue.

2. Introduction

Simply put, our purpose is to overview U.S. drug interdiction, particularly operations conducted in the Caribbean drug transit zone, highlighting the features most relevant to the potential application of Computer Generated Force (CGF) technology. If the issues are divided into two broad categories, those relating to interdiction operations and those relating to CGF development, our discussion is primarily in the former. Much of our description of the mission is

conveyed via a representative drug trafficking event, in this case an air transportation incident spanning the Caribbean. We selected this discussion vehicle, an actual event, to depict the complexity and challenges of the daily situation; an interagency force responding regionally within a very compressed timeline under restrictive rules of engagement. The foregoing statement alone implies the training and event analysis requirements that we believe offer the greatest payoff in the application of CGF technology to this problem. We use the mission, command relationships and infrastructure of Joint Interagency Task Force East in Key West, FL to represent the potential for CGF application in drug interdiction.

3. Just Another Night at the Office

It's very late in the evening on a Saturday and the atmosphere in the Joint Operations Command Center (JOCC) of the Joint Interagency Task Force East (JIATFE) in Key West, Florida is about to be dramatically transformed. Manned 24 hours a day, the JOCC is the focal point for coordinating the response of the U.S. government and its regional allies to air and maritime drug smuggling events as they occur in the Caribbean. Linked to military and law enforcement vessels, aircraft and radar installations ashore, JIATFE quarterbacks a diverse and far flung interagency and international team. Calling the play in the JOCC is the Command Duty Officer (CDO), on this particular evening an officer of the U.S. Customs Service (USCS), one of three U.S. agencies along with the Department of Defense (DoD) and the U.S. Coast Guard (USCG) that provide most of the 180 personnel who staff JIATFE. Little in the background of an officer from any of those three agencies prepares them thoroughly, prior to their assignment, for the challenge of orchestrating operations of a complexity and scale routinely experienced at JIATFE. Reflecting the interagency composition of JIATFE, the JOCC is continuously manned by a dozen personnel, specialists in both the Intelligence and Operations fields. Approximately half

of the crew is DoD with the remainder divided between the USCS and USCG.

The Caribbean area of responsibility (AOR) encompasses a region comparable in size to a triangle bounded by the cities of Miami-Seattle-New York and includes the territorial air and sea space of multiple nations whose cooperation with U.S. counterdrug operations covers the full range from strong to nonexistent. The assets coordinated by JIATFE to cover the AOR, limited in number but highly capable platforms, represent the principal agencies and nations engaged in interdiction. Constantly patrolling or on alert in the AOR is a drug interdiction force that includes:

- U.S. Navy vessels
- British and Dutch Navy vessels and aircraft
- Air Force fighter interceptors
- Airborne Early Warning aircraft (USAF E-3, USN E-2)
- Maritime Patrol aircraft (P-3)
- U.S. Coast Guard cutters and patrol boats
- U.S. Customs Service tracker aircraft
- Drug Enforcement Administration aircraft
- U.S. Army and Coast Guard helicopters (UH-60)
- DoD radar sites and associated command centers

And given the sensitivities of interagency operations in international waters and airspace, the JIATFE CDO is in a continual state of information exchange in a wide network that includes DoD, law enforcement and diplomatic agencies in places as diverse as Washington, Norfolk, El Paso, Colorado Springs, Puerto Rico, and American Embassies throughout the Caribbean.

The sequence of related events unfolding for our CDO actually began, fairly typically, long before his actual watch in the JOCC. Given the size of the AOR and the limited assets available, the difference between interdiction success and failure is often the degree to which pre-event intelligence from varied sources provides cues to decision-makers who play an educated guessing game in positioning the force to disrupt the traffickers' plan. Our Saturday evening operations can be traced back to the previous Tuesday. The intelligence community produced the indications of an impending transfer of some 400 kilograms of cocaine by a small, twin engine (propeller) aircraft flying from a remote airstrip on the north coast of South America to a rendezvous in either the Eastern Caribbean or the Bahamas. Upon reaching its destination the aircraft might land briefly and offload at a small airstrip or might drop its load in bundles to several small, "go fast" boats that will immediately scatter at high speed to isolated beaches and coves on the nearby islands. At cache sites ashore

the cocaine will be repackaged and later moved along the transportation pipeline to U.S. and European destinations for distribution. In moving contraband in this leapfrog fashion, the exposure of the shipment to interdiction is minimized and the drugs are conveniently warehoused along the way until needed at the distribution end of the chain.

While the intelligence preceding our Saturday evening event is invaluable, it is not sufficiently detailed for the JIATFE planners to commit assets to a narrowly defined course of action. Moreover, when the warning of the airdrop event was received, JIATFE was also planning against a reported surface transfer, potentially multi-ton in size, in the western Caribbean. The intelligence only indicates that the air event might possibly occur during a timeframe of several days and the location remains vague. With that level of foreknowledge, the interdiction plan places vessels and aircraft in positions from which they can flexibly respond to a number of potential airdrop sites while maintaining a realistic state of readiness for the projected period of time. Land based radar assets are oriented to enhance their surveillance of the projected flightpath. The interagency intelligence effort is focused on refining the initial warning. The JIATFE planners, their counterparts in other command centers, and the key personnel who will execute the operation, plan their roles in anticipation of foreseeable events and contingencies...and await developments.

Late Saturday the event begins to unfold with an initial detection of the suspected aircraft by the Remote Over The Horizon Radar (ROTHR) system that provides surveillance of the Caribbean and South America from bases in Virginia and Texas. The suspect aircraft is in a flight profile that matches that often used by drug traffickers. Thus begins the process to sort and identify that contact from the multitude of legitimate aircraft operating in the area. The earliest possible confirmation that the contact is the anticipated drug trafficker is necessary to give the surface and air units directly involved in the endgame the best opportunity to be in position to intercept the shipment.

Shortly after the initial ROTHR contact, the USNS Capable, a small specially equipped surveillance vessel (see the MOD T-AGOS description in Section 6) operating off the north coast of Venezuela, also detects the low flying, northbound aircraft and adds its track data to the overall detection and monitoring information flow into JIATFE. The CDO acts to obtain visual identification of the aircraft, now an Air Target of Interest (ATOI), and ensure that an airborne monitoring platform is in position to maintain contact with the ATOI for the duration of the event.

Two USAF F-16 fighters based in Puerto Rico are launched to intercept and covertly identify the ATOI. At about the same time a USN E-2C and USCS P-3 aircraft are launched to provide tracking and airborne command and control of the air assets that will converge on the scene. Following the F-16's identification of the aircraft and their return to base, a U.S. Customs Service tracker aircraft is integrated into the monitoring of the ATOI. Meanwhile a Surface Action Group consisting of a USCG Cutter and two USN Patrol Coastal vessels, with USCG boarding detachments aboard, is alerted to move to the vicinity of a possible airdrop. These surface assets, including the UH-60 helicopter aboard the USCGC, will attempt to disrupt the airdrop, block the escape of the high speed surface craft involved and recover any contraband floating in the area.

Our CDO, assisted by the intelligence analysts and operations specialists in the JOCC and in coordination with other regional command centers, is not only orchestrating the interdiction of the northbound movement of the ATOI, but also is acting to ensure that the aircraft is continually tracked on its southbound return home. At that point established information exchange procedures will be exercised to assist the host nation law enforcement agencies effect an arrest and seizure. The frigate USS Sides will be repositioned to assist in the monitoring.

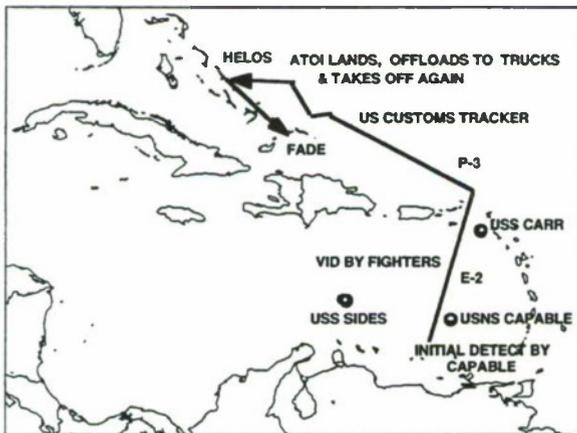


Figure 1. Eastern Caribbean Air Event

Meanwhile, instead of an airdrop our ATOI transits through the Eastern Caribbean, then turns to the northwest and ultimately lands on a small island in the Bahamas archipelago (see figure 1). In less than 5 minutes of ground time the cocaine is offloaded and the plane takes off again southbound. Minutes later two U.S. Army Blackhawk helicopters airlifting a joint arrest team of DEA agents and Bahamian police officers, arrive on scene from their base in the eastern Bahamas having been alerted and updated on the

developing event by JIATFE. The Blackhawks swoop into the airstrip vicinity and debark the arrest team while the narco ground crew is still present loading their cargo onto a truck. The overwhelming speed and force of their arrival discourages any hostilities and results in the seizure of the cargo, the vehicle and the arrest of three suspects. Unfortunately contact with the southbound ATOI is lost soon after its takeoff. Quite possibly it landed on another nearby island.

Nevertheless, a trafficking event has been frustrated. The shipment and some assets have been seized. Some arrests made. And equally significant, valuable investigative leads will be developed and followed up for the more proactive targeting, by U.S. and foreign law enforcement agencies, of the drug trafficking organizations' transportation, command and control and financial infrastructure. This event typifies transit zone activity. Known air and surface transportation events may exceed 100 during a given quarter. Any operation offers lessons to be learned and applied in the future. Our example is no exception. Although busy preparing for the next event while maintaining readiness, the participants consider the after-action issues:

- In reconstructing the event, how were the actions of the various elements integrated?
- How can the participating command centers and subordinate elements better prepare (train) for the event recognition, decision-making and coordination tasks that must be accomplished rapidly?
- Given the pre-event intelligence, was the preparatory force laydown optimal? What are the tradeoffs of alternative positioning schemes? Other interdiction assets?
- How does the course of action selected during the event compare with other alternatives?

These questions are constants for the interdiction planners and executors. Computer Generated Force (CGF) technology under development for the Synthetic Theater of War (STOW) offers a training, planning, and assessment tool, readily transferable from its conventional application to the interdiction aspect of the National Drug Control Strategy. An overview of the threat, interdiction strategy, and the supporting infrastructure will aid in understanding the requirement, potential application, and payoff.

4. Threat Overview

Although the flow of drugs from source to distribution can take several forms and routes, some geographic and tactical trends are evident. The principal air and maritime routes and the activity levels are as shown in figure 2.

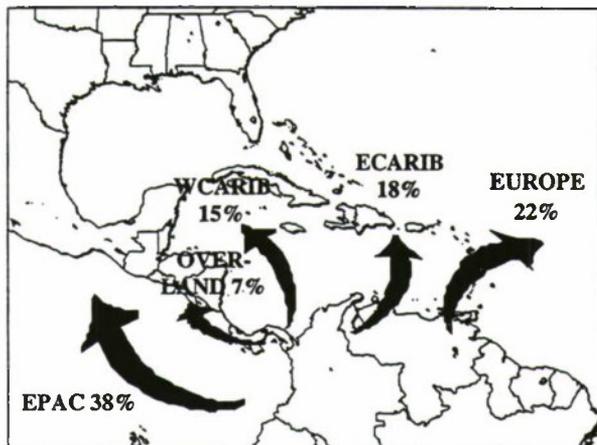


Figure 2. Principal Routes/Activity

Given their resources, drug traffickers have fielded an impressive array of air and surface platforms. Generally the vehicles of greatest interest (vulnerability) to the interdiction force fall into four categories:

- light twin-engine propeller or executive style jet aircraft
- large commercial style jet aircraft
- small, high speed surface craft
- small commercial (e.g. fishing) vessels

Traffickers often employ one or a combination of techniques to move their product through the transit zone.

- transport large shipments aboard commercial type jet aircraft (e.g. Boeing 727) non-stop from South America to remote airstrips in central/northern Mexico, flying routes and altitudes normally used by commercial air traffic in an effort to blend in with that flow
- transport smaller shipments aboard low flying propeller-driven aircraft from the northern coast of South America to rendezvous sites in the Caribbean islands where the aircraft either lands at a remote airstrip briefly to offload its cargo or drops the load in waterproofed bundles to waiting surface craft that move the cargo to cache sites ashore

- transport large shipments via surface platform (e.g. fishing, merchant vessel) from the coast of South America to an at sea rendezvous with smaller, high-speed craft that move the cargo to cache sites ashore
- transport smaller shipments via small, high speed surface craft from the coast of South America to at-sea rendezvous or cache sites in the islands
- transport shipment, concealed among difficult to search legitimate cargo, via surface merchant vessel directly into commercial ports

5. The Command and Control Structure

The U.S. Government's command and control system to combat the distribution of illegal drugs has evolved through many stages from its origins soon after the turn of the century. With each phase, the problem gained recognition as one of increasing severity and the number and diversity of agencies engaged in the effort steadily grew. In 1988, the Office of National Drug Control Policy (ONDCP) was created to bring unity to the activities of the numerous federal, state, and local counterdrug agencies. By law, the Director, ONDCP develops the annual National Drug Control Strategy (NDCS) that includes goals for the international interdiction program. The latest version of the NDCS (April 1996) places great emphasis on the interdiction component.

During the late 1980's, countering the production, trafficking, and use of drugs was declared to be a "high priority national security mission". With the passage of the FY 1989 National Defense Authorization Act (NDAA), Congress imposed specific new responsibilities upon the Department of Defense (DoD). Prior to this action, the DoD role had largely been to provide training and loan equipment to Drug Law Enforcement Agencies (DLEA). The 1989 NDAA significantly expanded the DoD role. The military was tasked to take the interagency lead in the detection and monitoring (D&M) of illegal drug shipments into the U.S., and was also tasked to create an integrated command, control, communications, and technical intelligence network linking the military and civilian agencies (See Figure 3). These new responsibilities were subsequently made part of permanent law in Title 10, USC.

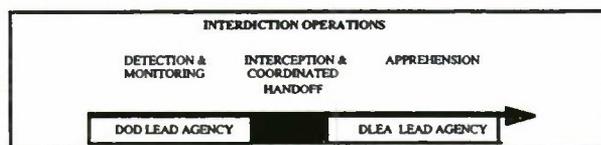


Figure 3. Interdiction Phases/Responsibilities

To accomplish this mission, DoD built upon the existing Unified Command structure. The Commanders-in-Chief of the Atlantic and Pacific Commands established Joint Task Forces Four and Five respectively in Key West, Florida and Alameda, California as operational command centers controlling D&M activity. Critical as they were, these two centers were only two nodes in a complex array of counterdrug operations and intelligence centers under both military and law enforcement leadership.

In 1994, Presidential Decision Directive (PDD) 14 directed that ONDCP "review the multiplicity of command and control and intelligence centers involved in international counternarcotics and recommend steps to streamline the structure". The result was the National Interdiction Command and Control Plan (NICCP). Figure 4 portrays the consolidation and agency relationships with respect to operations in the transit zone. The Commandant of the Coast Guard was empowered as the U.S. Interdiction Coordinator (USIC) with responsibility for coordinating the interdiction effort in the Western Hemisphere. The USIC is responsible for ensuring that assets for interdiction are sufficient and that their use is properly integrated. At the same time three Joint Interagency Task Forces (East, South and West) were established to provide command and control for the military and law enforcement assets assigned to interdiction. Though these command centers were built upon existing DoD infrastructure

including the former counterdrug Joint Task Forces Four and Five, the new concept integrated military and civilian personnel and assets under unified leadership to an unprecedented degree. The JIATF's are true national task forces comprised of U.S. Customs Service, U.S. Coast Guard, DoD and even allied resources.

Even with this trend toward interagency integration, the government's counterdrug program management remains exceptionally complex. The growing pains are not unfamiliar to those who've experienced DoD's jointness evolution. Some thirty federal agencies continue to exercise some form of drug law enforcement jurisdiction. Their members bring the diversity in policy, training, and procedures of their parent organizations to the interagency task forces. This coalition presents both opportunity and challenge to the Director, ONDCP and the USIC. The blend of the interagency ensures that the widest range of government talent and resources are continually engaged in the fight. The challenge, efficiently focusing the assets on specific objectives that contribute to the NDCS, is nowhere more evident than in the interdiction arena. And nowhere in the interdiction realm are the requirements, advantages, and limitations of interagency jointness better represented than in Joint Interagency Task Force East.

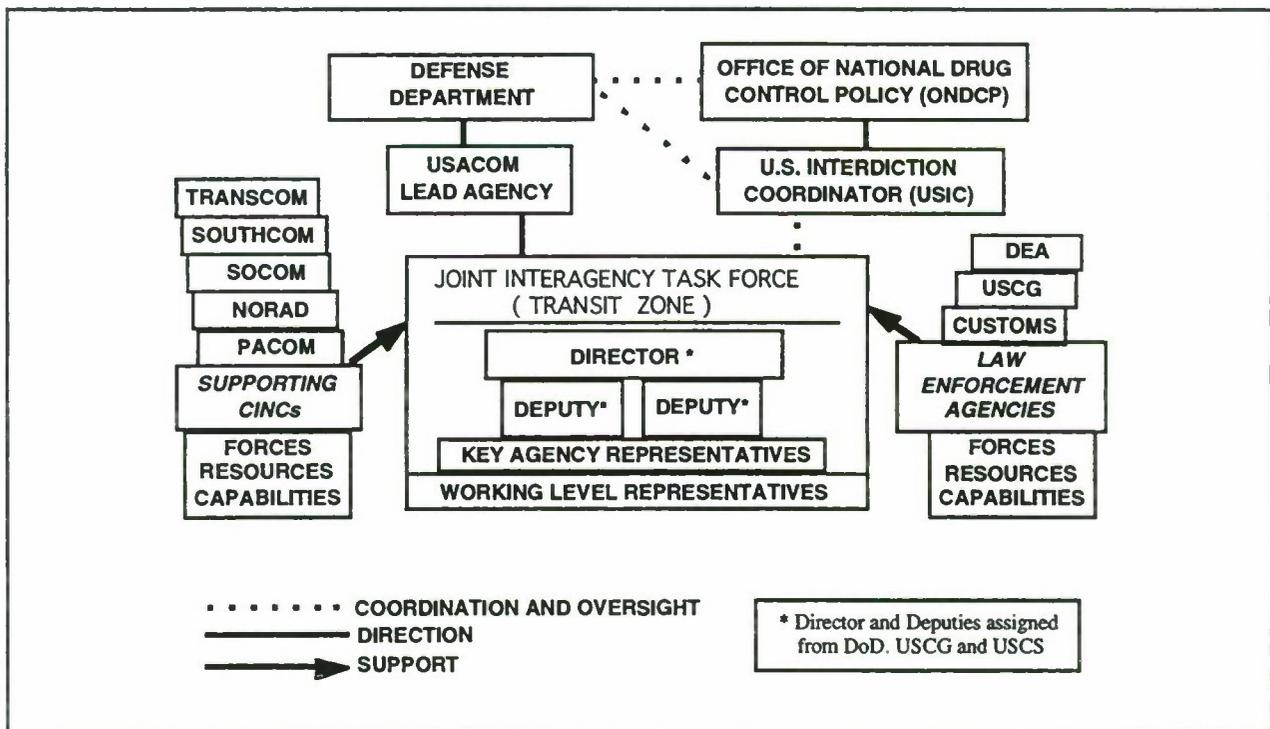


Figure 4. Command and Control Structure

6. Joint Interagency Task Force East

The scenario presented earlier illustrates the JIATFE mission, task force composition, and the interrelationships of the interdiction elements. Figure 5 depicts the task organized structure reflecting the U.S. and allied (Royal Netherlands Navy) elements. Although it's conceivable that virtually any air or maritime platform available in the DoD or drug law

enforcement inventories could be employed by JIATFE for specific purposes and periods the following descriptions highlight those most likely to be found operating in the AOR on any given day. The parent agencies are also indicated. Within DoD the assets are sourced from both active and National Guard forces and represent both the Atlantic and Pacific Fleets.

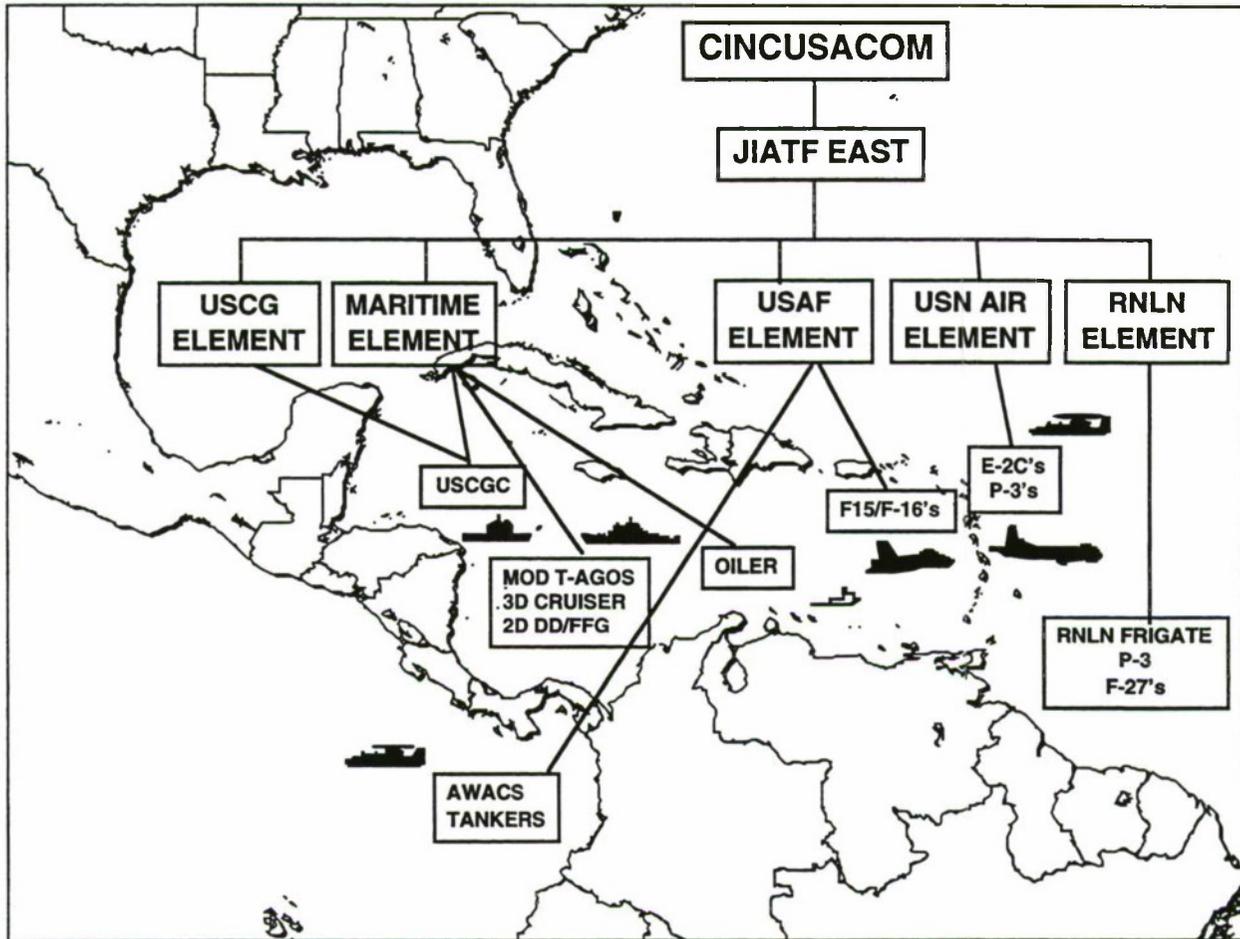


Figure 5. JIATFE Task Organization

Airborne Platforms. Airborne platforms provide counterdrug forces several capabilities. They can provide much greater height and range for electronic and visual search, reconnaissance, or surveillance missions. They provide a platform with equal or better performance than the drug smuggling aircraft to allow for interception and tracking, and they provide the means for DLEA officers to be rapidly deployed.

E-3 Sentry (AWACS) - (USAF). This is an airborne early warning, and command and control aircraft based on the Boeing 707 airframe. It is used for air

and maritime radar surveillance detection and tracking of suspected smuggler aircraft and vessels.

P-3 Orion - (USN, USCS). The Orion is a fixed-wing, multi-engine turboprop, Maritime Patrol Aircraft (MPA). It is used as a surveillance platform in the counterdrug role.

E-2 Hawkeye - (USN). This is a carrier capable, fixed-wing, twin turboprop, Airborne Early Warning (AEW) aircraft capable of detecting air and maritime targets.

F-15 Eagle / F-16 Fighting Falcon - (USAF/ANG). Single-seat fighter aircraft. Operated by the Air Force and the Air National Guard in the counterdrug role as interceptors.

UH-60 Blackhawk / Seahawk / Jayhawk - (USA/USN/USCG) This is a twin-turbine, combat assault transport helicopter. Operated in different variants by the Army, Navy, and Coast Guard, for surface search, airborne tracking, and DLEA apprehension.

Cessna Citation II - (USCS). The Citation is a modified twin turbofan, fixed-wing general aviation jet. It is equipped with an air search and tracking radar and FLIR. It is used by the Customs Service to intercept and track suspected smuggling aircraft.

Cheyenne III Customs High Endurance Tracker (CHET) - (USCS). The CHET is a modified twin turboprop, fixed-wing general aviation aircraft. It is equipped with radar, FLIR and VHF communications. It is used by the Customs Service to intercept and track suspected smuggler aircraft.

Afloat Platforms. Sea-based platforms provide counterdrug forces the advantages of mobility and high endurance. They operate in air and maritime D&M, interception, and apprehension roles.

High Endurance Cutters (WHEC) - (USCG). These 378 foot vessels are equipped with air and surface search radars and are capable of supporting a helicopter. They are used for air and maritime surveillance, interception, and apprehension.

Medium Endurance Cutters (WMEC) - (USCG). These 210 to 270 foot cutters are equipped with surface search radars and are capable of supporting a helicopter. They are used for maritime surveillance, interception, and apprehension.

Picket Ships - (USN). US Navy cruisers, destroyers and frigates are used as radar picket ships to provide air and maritime search and surveillance.

Modified Ocean Surveillance Ships (MOD T-AGOS) - (USNS). These are 224 foot ocean surveillance vessels capable of speeds of 11 knots and modified for counterdrug operations. They are equipped with an air search radar and are deployed in lieu of USN combatants. They are capable of data linking with other platforms and have extensive communications equipment.

Submarines - (USN). US nuclear powered submarines can provide information on both sea and air traffic while remaining completely covert.

Land Based Systems. Land based systems may be either fixed or mobile, depending on size and mission requirements.

Relocatable Over the Horizon Radar (ROTHR) - (USN). This is a Navy sponsored over-the-horizon backscatter radar system capable of providing wide area detection and surveillance of air targets up to 2000 NM from the site with real-time reporting of targets of interest via the Anti-Drug Network (ADNET) to appropriate agencies. There is currently one ROTHR site operating in Chesapeake, VA, with a second site in Texas. A third site is currently planned for installation in Puerto Rico.

Ground Mobile Radars - (USAF, USMC, ANG). These mobile radar sets provide primary or augment existing radar coverage and are capable of long range searches up to 240 nm, and height finding up to 95,000 ft.

Caribbean Basin Radar Network (CBRN). The CBRN is a series of linked U.S. and host nation radars throughout the Caribbean.

6.1 Concept of Operations

To tailor the concept of operations (CONOPS), JIATFE uses a planning cycle which considers: the threat, asset requirements, asset availability, and both pre-planned and quick response operations. JIATFE publishes periodic threat assessments that are sent to all headquarters and agencies that provide D&M asset support. With that as a basis, JIATFE hosts regional planning conferences where a CONOPS for an upcoming period is developed. JIATFE then publishes the CONOPS for execution.

JIATFE's operational concept is built on defense in depth to detect and monitor drug traffickers as close to the source country as possible, followed by continuous monitoring using a mixture of electronic and visual means as the target transits across the AOR, and finally handing off the target to DLEAs. The process is extremely complex because it frequently involves several military commands and Federal agencies. To accomplish this, JIATFE employs a mixture of DoD and DLEA assets and sensors to conduct routine patrol operations and respond to changing intelligence assessments. The actual employment of ships and airborne assets is determined on a daily basis in response to current intelligence information concerning ongoing or expected drug trafficking operations. Assets are positioned to optimize time-on-station to cover threat routes. Timely intelligence support enables JIATFE to provide target alerts to law enforcement command centers allowing cueing of assets for successful

apprehensions. Once an aircraft has been detected and sorted by JIATFE, it is monitored in transit until a positive handoff or other disposition is coordinated for apprehension by the DLEAs. Maritime targets are handled in much the same manner.

6.2 Connectivity

As touched upon previously, one of the principal DoD counterdrug responsibilities under current law is to integrate counterdrug "command, control, communications, computers, and technical intelligence (C4I) assets of the US" into a

communications network. The Defense Information Systems Agency (DISA) is responsible for the integration of the national telecommunications and information systems master plan for the Federal DLEAs. The backbone for counterdrug connectivity is the Anti-Drug Network (ADNET). The ADNET provides rapid, secure, and interoperable C4I connectivity supporting the counterdrug missions for both DoD and non-DoD agencies (Figure 6). ADNET uses the Joint Visually Integrated Display System (JVIDS) as the primary means to exchange and display information over a primary framework provided by Defense Data Network (DDN).

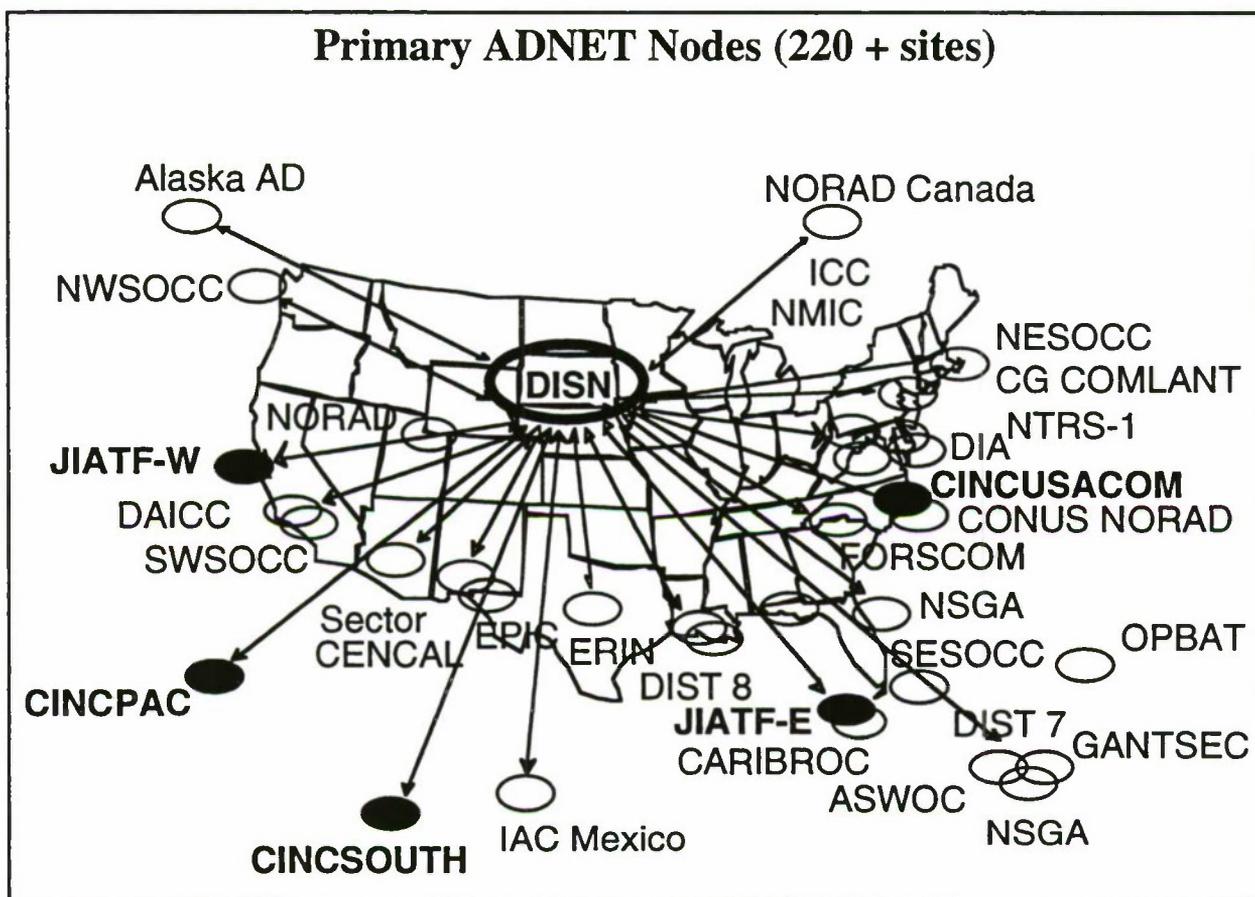


Figure 6. The AntiDrug Network (ADNET)

ADNET nodes are subdivided into command, operational, and intelligence sites. The command sites exercise oversight responsibility for counterdrug operations. Operational sites are charged with command and control over intelligence gathering assets, D&M and/or suspect target interdiction. Intelligence sites are primarily involved with fusing, analyzing, and dissemination of information within ADNET. Figures 7-9 are an overview of counterdrug connectivity in the Caribbean zone including the

circuits, integration of ADNET with tactical systems i.e., Tactical Data Information Link (TADIL) and Link 11, and the Officer in Tactical Command Information Exchange System (OTCIXS).

Path	Circuit	To		Comment
T1 satellite	ROTHR (data)	Northwest	2400 Baud	Raw Radar Data
	ROTHR (voice)	NW/ CARIBROC	Voice	Voice Coordination
	CASREP/SORTS	CINCLANTFLT	2400 Baud	
	Phone (x 5)	USCINCLANT	Voice	
	JWICS (SCI data)	Interagency	384K	SCI Data
UHF satellite	VTC	Interagency	384K	SCI VTC
	TRE	NSGA Key West	2400 Baud	ELINT Data
	OTCIXS	TG 4.1/Other	2400 Baud	C2
	101	Interagency	Voice	JIATF-E NECOS
	401	TG 4.1	Voice	Admin/Logistics
	402	USCG	Voice	Able Manor Net
	403	TG 4.1	Voice	AW NECOS
	407	Cryptologists	SI Voice	
	409	Selected	Voice	Restricted Ops
	Andean Ridge	JIATF South	Voice	JIATF South C2 Net
SIPRNET	ADNET	ADNET	56K	Geneser Data
	NTRS	NSGA KW	56K	
	INTELINK-S	Interagency	56K	(ADNETLINK)

Figure 7. JIATFE Circuits

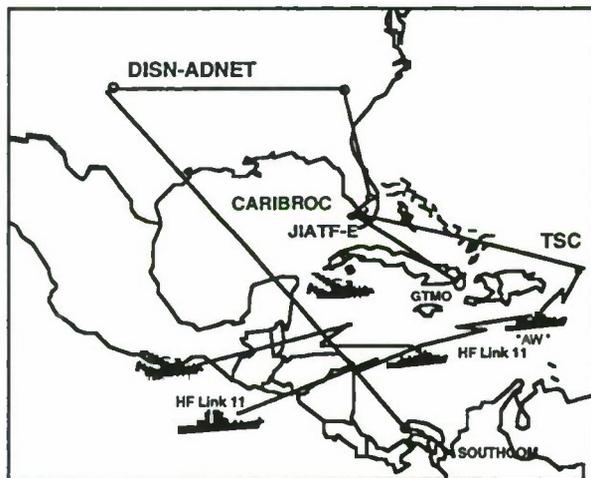


Figure 8. Tactical Data Information Link (TADIL) A/Link 11 Integration

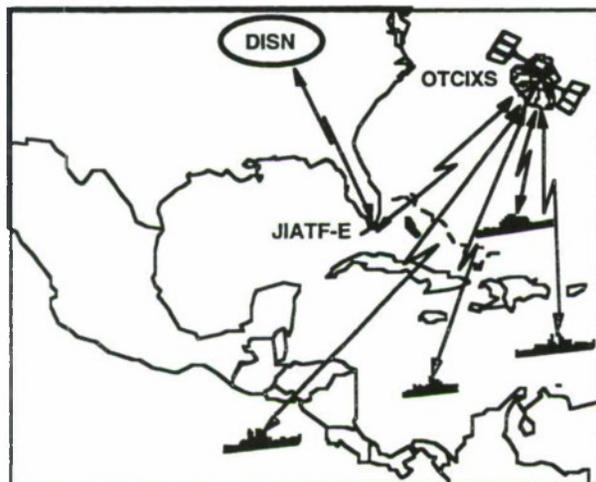


Figure 9. OTCIXS to ADNET Bridge

6.3 Training

We've already seen that the interagency team-building responsibility of the JIATFE Director is significant. Task force members manning the command center and other critical nodes bring their diverse backgrounds to a process that must produce and maintain the skills required to master the crisis action essential to interdiction command and control. CGF technology distributed via realistic force representation to the key nodes of the counterdrug command and control network (e.g. JIATFE, land based radar command and control sites, ships) is a vehicle for molding interagency teams in the same way that DoD is pursuing training at the Joint Task Force level. Analyzing the training needs of the JIATFE level training audience and their counterparts points to some general requirements:

- Monitoring the battlespace
- Threat assessment
- Developing plans/taskings for the JIATFE elements
- Managing information flow and reporting
- Familiarity with platform and system capabilities

Strategic Theater	
ST 8	Develop and Maintain Alliance and Regional Relations
ST 8.4	Provide Theater Support to Other DoD and Government Agencies
ST 8.4.1	Support Counterdrug Operations in Theater
Operational	
OP2	Develop Operational Intelligence
OP2.2	Collect Operational Intelligence
OP2.2.2	Collect Information on Operational Targets
OP2.2.3	Provide Operational Reconnaissance and Surveillance
OP2.3	Process Operational Information
OP2.3.2	Analyze and Evaluate Operational Areas
OP2.3.3	Integrate Operational Intelligence
OPI	Conduct Operational Movement and Maneuver
OPI.2.2	Posture Joint Forces for Operational Formations
OP1.2.4	Conduct Operations in Depth
OP1.5.4	Isolate Theater of Operations
OP5	Exercise Operational Command and Control
OP5.1.1	Communicate Operational Information
OP5.1.3	Maintain Operational Information and Force Status
OP5.3.7	Select or Modify Course of Action
OP5.4.4	Synchronize/Integrate Operations
OP5.7	Coordinate and Integrate Joint/Multinational and Interagency Support
OP5.7.4	Coordinate Plans with Non-DoD Organizations

Figure 10. Mission Essential Task List

Although it's not our intent to develop an all inclusive counterdrug mission essential task list, DoD's Universal Joint Task List (UJTL) provides a useful framework. The UJTL is a comprehensive hierarchical listing of the tasks that can be performed by a joint force. It is organized by the level or echelon of the activity, Strategic National (SN), Strategic Theater (ST), Operational (OP) and Tactical (TA). The Strategic Theater and Operational tasks in figure 10 have been extracted from the UJTL as a possible list of priorities for counterdrug purposes at the JIATFE level.

6.4 Event Analysis

The interagency counterdrug community is in a continual state of event analysis, dissecting individual trafficking events and identifying trends for the information that will refine interdiction tactics and focus limited assets for the greatest payoff. Considerable effort and expense is rightfully devoted to this function. The analysis can be further distinguished as either pre-event planning or post-event assessment. As we saw in our introductory scenario, pre-event intelligence cueing is the foundation of an effective response. Situationally dependent, the quality of intelligence and the time remaining may allow for detailed planning, comparison of courses of action and consultation with all key participants. In that case, the capability to distribute the anticipated event and alternative responses to the principal counterdrug C2 elements using representative CGFs would be an invaluable planning and mission rehearsal enhancement. This "drawing of the play in the dirt" with realistic CGF tools would not only strengthen common understanding of the plan but would also identify force deficiencies. Post-event assessments can range from the relatively informal reconstruction in the immediate aftermath to the more structured process of the Interagency Counterdrug Performance Assessment Working Group (ICPAWG). The ICPAWG was established in 1992 to develop a data base of known drug smuggling activity and to measure interdiction performance. The process consumes the effort of some 35 representatives of the principal U.S. counterdrug and intelligence agencies as they review in great detail the prosecution of each event. This necessary review reduces duplicative agency reporting and provides a current threat basis for future planning and asset allocation.

7. Computer Generated Interdiction Forces

CGF technologies provide the opportunity for interoperability of constructive simulation programs, virtual forces, and instrumented live play facilities, which can provide for an interdiction training

environment as good or better than "real-world" on-the-job training. The U.S. Navy's experience and program goals seem particularly appropriate for the similarities between the Navy's conventional air and maritime interdiction mission and the requirements of JIATF-led drug interdiction operations. We've seen that the Navy already contributes a substantial portion of the DoD resources dedicated to transit zone interdiction. The Navy has also amassed a talent pool in the application of CGF simulating those same air and surface platforms. Much of that experience is fairly recent. Prominent milestones include the Synthetic Theater of War-Europe (STOW-E) Technical Demonstration, during November, 1994 and Exercise Kernel Blitz 95 (KB95), during March and April, 1995.

STOW-E was significant in that it served to indicate that modeling and simulation technologies had the potential to contribute to operational training. The demonstration showed that synthetic forces could be integrated with live forces to permit exercise participants to train in a seamless "theater of war" that more accurately represents real-world operations. KB95 was the Navy's first opportunity to integrate synthetic forces into an operational exercise. KB95 presented an overall scenario that linked several smaller exercises in the Southern California and Gulf of Mexico Areas. The simulation support task in KB95 was to enrich the exercise by simulating a Carrier Battlegroup and opposition forces for the "live" Amphibious Task Force engaged in the exercise. The Navy tapped the capabilities of modern simulation laboratories and training facilities for KB95. Analysis of KB95 clearly showed that the training accomplished in a STOW environment can increase readiness by enriching the training environment and present additional, more complex training opportunities to exercise participants. KB95 showed that the use of CGFs offers a chance to conduct training that otherwise may not be possible because of budgetary or other restrictions.

The core simulation for KB95 relied on geographically dispersed simulations linked via the Defense Simulation Internet (DSI) communicating among one another using 2.0.3 Institute of Electrical and Electronics Engineers (IEEE) standard Distributed Interactive Simulation (DIS) protocols. The primary simulation engine was CGFs in the form of Loral/Defense Advanced Research Projects Agency (DARPA) Modular Semi-Automated Forces (ModSAF) and the Battle Force Tactical Training (BFTT) Program Operational Procedures Consoles. The ModSAF used was a variant of the Loral ModSAF Version 1.4 modified by DARPA for generation of Navy, Air Force, and Opposition Force (OPFOR) aircraft in the What If Simulation System

for Advanced Research and Development (WISSARD) facility located at Naval Air Station Oceana, VA. ModSAF blue air forces were generated from Fleet Combat Training Center, Pacific, San Diego, CA. and OPFOR air from WISSARD. Man-in-the-loop tactical submarine simulators physically located in San Diego, CA and Groton, CT were also used. Man-in-the-loop mock-ups were configured with Link 11 and OTCIXS capability. All simulations were integrated using the DSI and commercial lines.

A review of the technical accomplishments of KB95 showed that exercises can be significantly enhanced through the use of CGFs and that distributed and interactive CGF simulations can be effectively interfaced with real-world C4I systems for training purposes. Borrowing upon the practical experience gained in supporting both STOW-E and KB95, a series of demonstrations of CGF capabilities were conducted for representatives of both the USACOM Counterdrug Operations Division and the JIATFE during the October 95-February 96 period at WISSARD.

The demonstrations were conducted on an independent (nonet) pocket system using a variant of Loral ModSAF modified by DARPA. The ModSAF platforms which were chosen to portray the various

aircraft and ships were already similar enough in system and behavioral capabilities that minimal modification was required (e.g. top end speed) to replicate the drug-running ("Go-Fast") boats. For presentation purposes the icons presented on the Graphical User Interface (GUI) were changed to look like the appropriate platforms. Following an initial "canned" scenario a number of other typical counterdrug events were initiated and the observers were allowed to participate in order to "drive" the outcome of events with decisions similar to what might be expected of a CDO on watch in the JIATFE JOCC. The demonstrations were typical of the kind of event driven exercises which could run concurrent with or independent of actual JIATFE operations, and served to show that the complexity of the JIATFE organization and operation lends well to a varied mix of constructive, virtual, and live exercise participation. The JIATFE JOCC is an outstanding setting for an instrumented live play facility, as would be select land-based radar sites at various locations in the U.S. and the Caribbean. Aircraft simulators i.e., an AWACS and/or E-2 simulator, could provide "virtual" detection and monitoring support of exercise operations. And, to further enhance training, constructive CGFs (friendly and opposition airborne and surface platforms generated at sites like WISSARD) can play an active part in interdiction exercises.

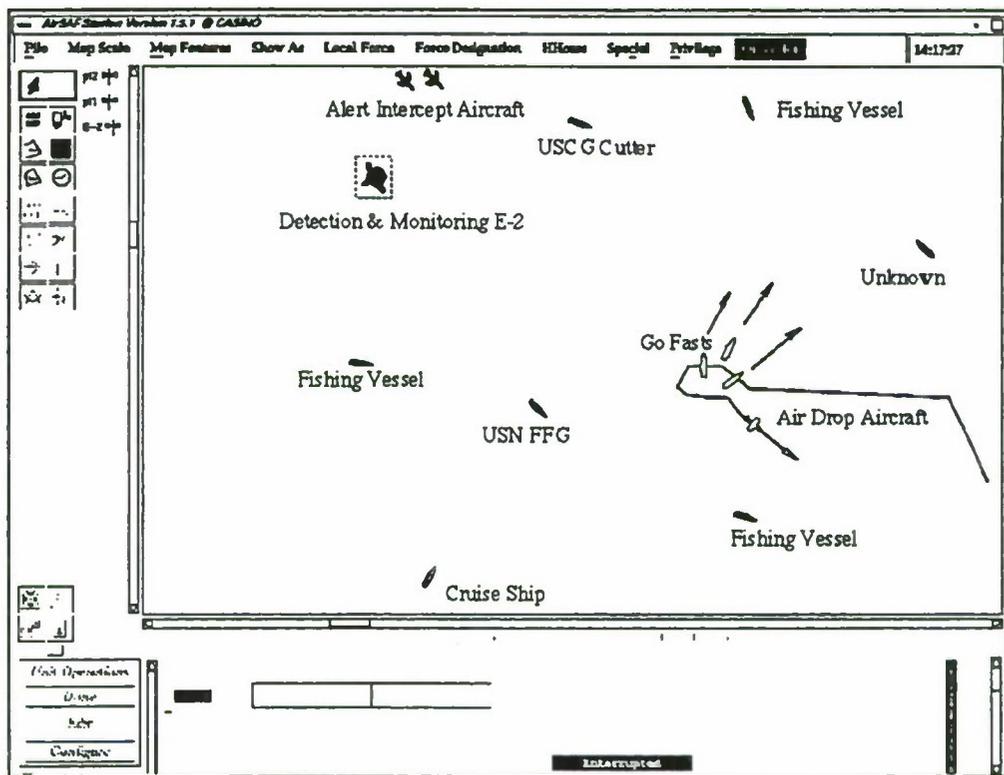


Figure 11. Interdiction Demo GUI Display (WISSARD Lab)

8. Summary

Not surprisingly this conceptual paper raises a number of implementation issues that require further investigation. We recognize the need to refine the training and analysis applications, training audience and objectives, network and other considerations. Planning for the CGF distributed simulation demonstrated in the Kernel Blitz exercise previously cited was considerable. Clearly we believe that there are sound reasons for continued study of the counterdrug application of CGF concept. Prominent among them are:

- The systems and behaviors of the most capable air and surface platforms commonly found in the interdiction force inventory match closely those that are currently in development as CGF entities (particularly Navy and Air Synthetic Forces). Representative "OPFOR" are also within reach.
- The interagency interdiction task force mission inherently lends itself to the use of CGF in distributed interactive simulation. Implicit in the mission is the need to continue operations on a 24 hour basis while training and integrating a diverse interagency team.
- As components of the principal DoD counterdrug commands (USACOM, SOUTHCOM and PACOM), the JIATFs are well positioned to transfer DoD expertise in the application of CGF technology to the interagency. In particular, JIATFE, a component of USACOM, is well situated to reap the training and analysis benefits of STOW technology and the emerging capability of USACOM's Joint Training Analysis and Simulation Center (JTASC).
- Conversely, the counterdrug arena represents a ready operational forum providing feedback to the CGF development process.
- The experience gained in air and maritime drug interdiction can be applied in other missions tasked in recent years (e.g., enforcement of embargoes, no-fly zones, etc.).
- The counterdrug C4I network, also a DoD area of expertise and responsibility, is in place as a framework which can support the application of distributed simulation for counterdrug operations/exercises.

The current investment in drug interdiction is substantial. At the federal level \$1.4B of the annual national counterdrug budget of \$15B is aimed at

interdiction programs. Detection and monitoring operations consume \$226M of which \$143M supports the operations we've described in the JIATFE AOR. This is a level of effort and a national priority worthy of the readily available enhancement resident in CGF technology.

9. Acknowledgment

We are deeply grateful for the assistance of the military and civilian personnel who unselfishly gave time from their already taxed schedules to guide this work. Notable representatives include Cdr Faris Farwell (USN), Maj Tony Crowder (USAF), Mr. Shane Hoffman, and SSGT Ron Talley (USAF) of the USACOM counterdrug operations and intelligence divisions.

10. References

Chairman of the Joint Chiefs of Staff (1994)
Joint Pub 3-07.4, Joint Counterdrug Ops
Chairman of the Joint Chiefs of Staff (1995)
CJCS Manual 3500.04, UJTL
Commander Naval Doctrine Command (1995)
Kernel Blitz '95 M & S Final Report

11. Author Biographies

John Miller and **Greg Jackson** are Systems Engineers employed by BMH Associates, Inc. They perform Knowledge Acquisition / Engineering in support of the development of USMC Ground and Air Synthetic Forces for STOW under DARPA sponsorship. Both have an operational background in the U.S. Marine Corps and in the counterdrug interagency.

Will Miller (no relation) is employed in the Programs, Resources and Technology Office of the Joint Interagency Task Force East. Mr. Miller has an extensive background in counterdrug programs at the national and regional levels.

Session 2a: Reasoning

Coradeschi, Linkoping University, Sweden

Stytz, USAF AFIT

Jones, University of Michigan, AI Lab

Kocabas, Marmara Research Center, Turkey



Intelligent Agents for Aircraft Combat Simulation

Silvia Coradeschi, Lars Karlsson, Anders Törne
Department of Computer and Information Science
Linköping University
581 83 Linköping, Sweden
{silco larka andto}@ida.liu.se

1. Abstract

In this paper we give a first account of a project at Linköping University in collaboration with Saab Military Aircraft AB, Sweden. The aim of the project is to study the design and implementation of intelligent agents for the air combat domain, especially for beyond visual range combat. In particular users should be able to construct such agents without computer science expertise. Our main interest lies in the decision process of the agent and in how the behavior of the agent can be specified.

2. Introduction

Systems for simulation of beyond visual range combat are currently used for evaluating aircraft, missiles and tactics and also for training pilots and to supply enemy aircraft in battlefield simulations. These applications require that the behavior of the automated pilots is almost indistinguishable from human behavior in the specific air-combat domain. This is especially relevant in training applications where humans interact with automated pilots. It is, in fact, important for the realism of the simulation that humans interact with automated agents as they would interact with other humans.

Human pilots are in general guided in their actions by strategies and tactics, but are also able to react to unexpected situations and to adapt general strategies to specific cases. The challenge is then to develop the capability in an intelligent agent to react in a flexible way to uncertain and dynamic environments and still follow strategies and tactics as a human pilot would.

Our interest is focused on the decision process of the automated pilot. An approach to modeling the human decision processes is proposed in TAC BRAWLER, a simulation tool for providing a detailed representation of air-to-air combat, both within and beyond visual range (Decision-Science Application 1991). In TAC BRAWLER the decisions about what actions to perform are made by the automated pilots which evaluate, for each alternative

action, the situations that will result if the alternative were to be executed.

The TacAir-Soar is also an interesting approach to air-combat simulation (Tambe et. al. 1995a). TacAir-Soar has been developed within Soar, a software architecture created as a basis for general intelligence. In this project several features that are required for an intelligent agent have been considered, for example coordinated behavior (Laird et. al. 1994) and enemy tracking (Tambe et. al. 1995b).

In both these systems air-combat experts can specify the behavior of the agents, but it is then encoded by system experts. In our approach the experts of the air-combat domain are intended to specify the behavior of each agent directly, without the aid of a system expert. This has the advantage of letting the experts give the directives that the agents should follow, test the behavior and change the directives in case the agents do not behave as expected. It also makes it possible to tailor the behavior for testing specific features of aircraft, missiles and tactics or to train pilots for a particular situation. On the other hand this puts special requirements on the user aspects of the system, both with respect to learning it and working with it. These aspects are considered in TACSI (TACTical Simulation) (Saab Military Aircraft 1995), a system developed by Saab Military Aircraft for autonomous simulation of many vs. many beyond visual range combat. This system is currently used at Saab for evaluating and developing their products. In parallel to the continued development of TACSI, Saab and Linköping University have undertaken a collaborative project with the aim of further investigating the design and implementation of automated agents for the air combat domain.

In our approach a scenario contains the specification of the agents present in the scenario, a state for each agent describing the information that is used for making decisions, and a decision-tree for each agent. So far we have implemented a prototype of the decision-mechanism and we have interfaced it with a simplified simulator, where the technical characteristics of aircraft and missiles are described though not yet on a high level of accuracy. The next step will be to design a user-interface for specifying the decision-trees, interface the decision system with the simulator currently used at Saab Military Aircraft and test the system with respect to both the performance and the user issues.

In the next section we see how the requirement that the user defines the decision-tree has influenced our choices in building the system. We then examine specification of the agents, state, decision-tree and actions and finally we consider an example.

3. Requirement for user-defined decision-trees

The knowledge required by an agent to perform realistically in complex and uncertain situations is in general difficult to acquire and to code precisely by the persons who implement the agents. Further-more the user can require different behaviors of the agents for different purposes, for example a pilot's trainer may want a very simple model of the behavior of the agent in order to train an inexperienced pilot. It then seems reasonable to give the user the possibility to determine the behavior of the agents and to adapt it to their particular context.

The challenge is to design a mechanism for specifying the behavior of the agent that is powerful enough for the user to specify the desired behavior within reasonable limits, but at the same time is easy to learn and to use.

3.1 Hierarchical structure and language of conditions

Specifying the decision mechanism in the form of rules is a natural way to represent behavior. However, the number of rules and conditions necessary for specifying complex behaviors can be very large. TacAir-Soar for example contains about 1700 rules. Also several conditions can be common to more than one rule. Therefore organizing the rules in a tree can contribute to a more structured and compact representation and this can facilitate the writing of the rules and the understanding of the resulting behavior. In our solution each node in the tree is associated with a condition that has to be satisfied in order to enter the node, and each "end node", or leaf, is in addition associated with an action. A branch of the tree represents a rule that has as conditions the conjunction of all the conditions in the arcs of the branch, and as action the action in the leaf. In this way the conditions are structured from more general to more specific and can be common to several branches yielding a more compact representation.

It is also important to allow the user to write complex conditions. For example a condition for performing the action of moving toward an enemy in order to intercept him can require that the enemy in question has been explicitly selected for interception and that the agent knows his position. A language has been defined for writing conditions that allow the use of *and*, *or*, *not*, *there-is*, *for-all* and functions.

3.2 Priorities and changes of priorities for actions

Several branches of the tree can be visited in parallel. If a leaf is reached, the corresponding action is a candidate to be performed. All the candidate actions form the candidate action set. From this set, the actions that are actually performed are selected on the basis of dynamically varying priorities. The user specifies a priority value for the action in each leaf and also specifies how this value can be changed during the evaluation of the branch conditions in which the action is a leaf. We have introduced the possibility of dynamic change of priorities in order to simplify the tree and to simulate the change of priority, in the pilot's mind, of the actions that he should perform depending on the situation.

The following example illustrates how dynamic change of priorities is used. In figure 1 a part of a decision-tree is presented. The whole tree is presented and explained in section 7.

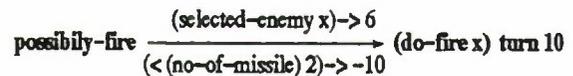


Figure 1: Example of dynamic change of priorities

The agent has the possibility to fire at an opposing agent x and should decide whether to fire or not. If x is the selected enemy, it should fire. If is not the selected enemy, it should fire only if more than 1 missile is left. In the example, if x is the selected enemy the priority of the sequential action (do-fire x) turn is increased by 6, and if the number of missiles is less than 2, the priority is decreased by 10. Given that the priority of the action is 10 and that the action is not performed if its priority is equal to or less than 0, the behavior is the desired one. In order to implement the same behavior without a dynamic change of priorities four action nodes would be necessary (figure 2).

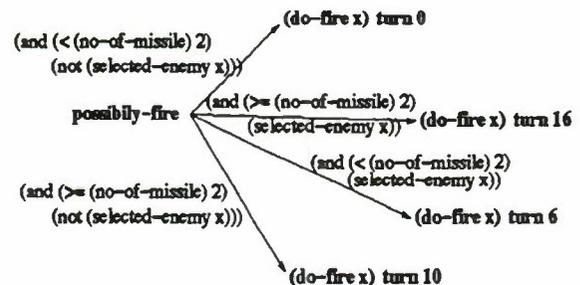


Figure 2: The same example as the previous figure without dynamic change of priorities

The actions at the leaves of the decision-tree can also be concurrent and sequential actions. Sequences of actions are useful if the user wants an action to be followed by other actions, for example firing and turning in the previous example. Concurrent actions are used when a number of actions should be started at the same time, for example turning and releasing a decoy in case a missile warning is received.

4. Specification of the agent

The specification of the agent contains the characteristics specific to each agent. These characteristics are specified by the user before starting the simulation and are maintained in the state of the agent. Examples of characteristics are:

- Physical characteristics, such as type of aircraft, number and type of missiles and amount of fuel.
- The role of the agent in the simulation, for example leader or wingman.
- A description of the mission that the agent should perform.
- General criteria the agent should follow in performing actions, for example commit criteria, i.e., criteria for deciding whether to intercept an enemy aircraft, and rules of engagement, i.e., directives about general conduct during the mission.

5. State

Each of the agents has a state where the information is maintained that is necessary for deciding what actions to perform. This information is of four types. First, the state contains the characteristics of the agent that the user has specified.

Secondly there is the information the agent receives from the simulator: position, velocity and direction of other aircraft visible on the radar and missile warnings. This is the same information that a pilot would receive from on-board sensors.

Thirdly, there is the information about the present status of the agent, the direction, velocity and position of the aircraft, the number of missiles left, friendly aircraft still present and the actions currently being performed, for example the agent is following the leader or moving toward a point.

Finally, one part of the state represents the "memory" of the agent. In the memory are recorded, among other things, important past events, for example at whom the agent has fired and how long before, and decisions previously taken, for example the decision

to select an enemy as main target or the decision to intercept an enemy, and orders received via communication channels.

6. The decision-tree

The decision-tree consists of a hierarchy of decisions with one decision for each node. At every cycle in the simulation the decision-tree is visited and a list



Figure 3: Phases of the decision mechanism

of possible actions is created (figure 3). The actions in the list are the actions that the agent will consider performing. To this list the actions still in progress are added. The list is sorted in order of priority, and the agent picks actions from the top and downwards, testing that each action is compatible with the higher prioritized actions already chosen.

6.1 Decision-tree structure

In a decision-tree, figure 4, a node is entered if the entry condition associated with it is satisfied. In the leaves there are actions with a basic priority value. Modifier conditions can also be associated with a node. Modifier conditions have the form (condition → number). If the condition is true, the number is added to the priority of the actions associated with the branches. If the priority of the action falls to 0 or below, the action is not performed.

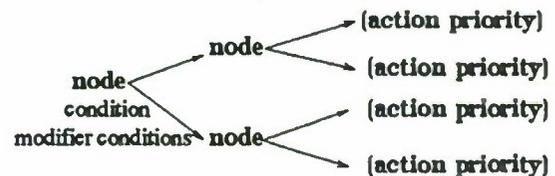


Figure 4: Decision-tree structure

6.2 Conditions

The conditions are defined as follows:

- Primitive Conditions

Atomic Conditions

Atomic conditions are true if the information is present in the state, for example the atomic condition

missile-warning is true if in the state the information that the agent has received a missile warning is present. Atomic conditions can also have the form "(property agent)" and in this case a check is made in the state to establish whether the property is satisfied by the agent. For example (*selected-enemy x*) checks if x is the selected enemy of the agent. The variables can only refer to agents in the present implementation, but the system can be easily extended to allow other type of variables too.

Relational Conditions

Relational conditions consist of relational operators such as $<$, $>$, \leq , \geq , \neq , $=$ applied to numbers and numerical functions. An example of a relational condition is ($<$ (*distance-to x*) 100) where (*distance-to x*) is a function that returns the distance of the agent to another agent x.

- Composite Conditions

Composite conditions are formed by applying the operators *and*, *or*, *not*, *for-all* and *there-is* to other composite or primitive conditions. If the condition associated with a node has the form "(for-all x condition)" the condition is tested for each agent x present in the simulation. The rest of the branch is visited once for each of the agents for which the condition is satisfied. All the actions selected are then taken into account as candidate actions to be performed. A "(there-is x condition)" finds the first agent that satisfies the condition and continues to evaluate the tree with x as this agent.

6.3 Actions

There are three kinds of actions: primitive actions, concurrent actions and sequential actions. Primitive actions are, for example, (*do-fire x*) i.e. fire a missile

at the agent x. They can also be internal actions, i.e., actions that update the state of the agent. For example (*selection-enemy x*) records in the state the information that x is the current selected enemy of the agent. The actions sent to the simulator can be instantaneous, for example (*do-fire x*), or can have a duration such as turn. The agent keeps track of the actions that he is currently performing in the state and takes this information into account when deciding what actions to perform next.

Concurrent actions consist of a collection of actions that should be started at the same time. There is a concurrence of actions inherent in the system as actions in different branches can be selected and, if they are compatible, started at the same time. As for explicitly concurrent actions, however, we are sure that either all the actions are performed or none of them are. An example of a concurrent action of the latter kind is the sending of a command from the leader to the wingman to select a specific aircraft as primary target and the internal action of recording that the wingman now has a selected target.

Sequential actions are composed by actions that should be performed one after another. When a sequential action is started, the first action of the sequence is started and an automatic record is made in the state as to which are the following actions in the sequence that should be performed. When the first action of the sequence is completed, the second action of the sequence becomes a candidate action to be started and it competes for starting with the actions currently being performed and the other actions selected at the present cycle of the simulation. If the second action is started, the same thing then done with the third action and so on until the sequence is completed.

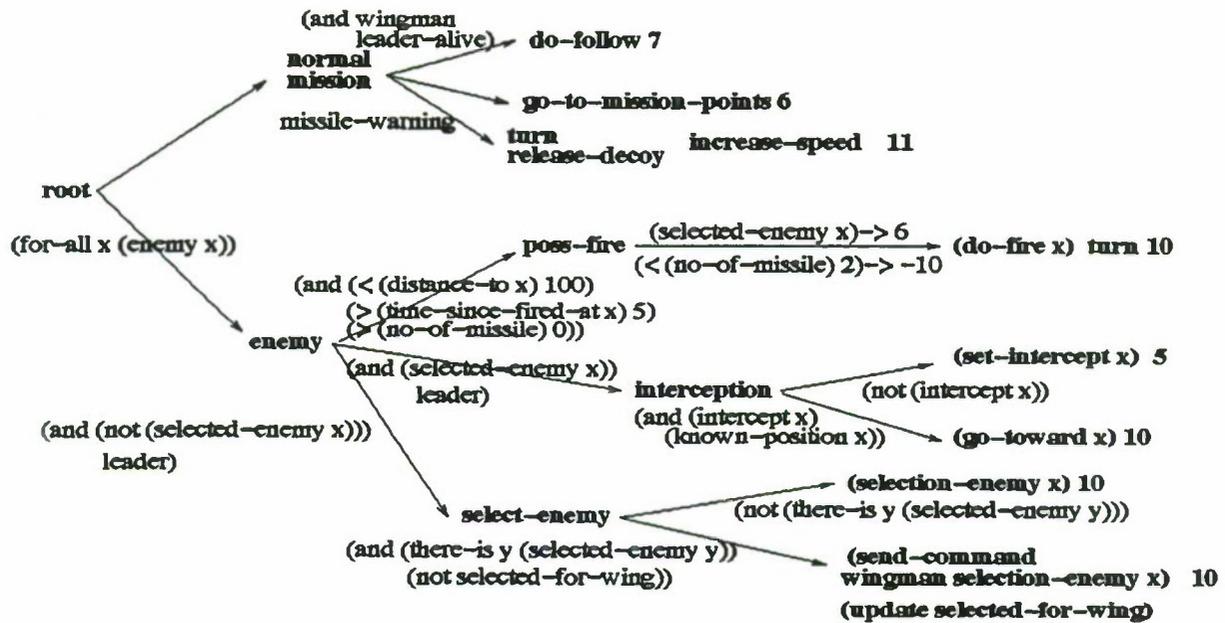


Figure 5: Example of decision-tree

If an action that is in turn is not started, the sequence is aborted. When a sequential action is started, the priority of the following actions in the sequence is increased with a value decided by the user depending on how important it is to complete the sequence once it has been started.

Finally actions can be a composition of both concurrent and sequential actions. For example, in the case of a missile warning the aircraft turns and at the same time releases a decoy and then it increases its speed.

6.4 Compatibility of actions

The test to check if two actions are compatible is made on the basis of the physical resources they use. Two primitive actions are compatible if they do not use the same resources. For example the action of firing is compatible with the action of turning left as they use different resources. A concurrent action is compatible with another action if all the actions that form the concurrent action are compatible with the other action. A sequential action is compatible with another action if all the actions that are present in the sequence are compatible with the other action. In this way we consider incompatible actions that could be performed at the same time. For example, the action of firing can be performed at the same time of the sequence of actions of turning and firing, but in order to build a compatibility criterion that would

take these cases into consideration, it would be necessary to predefine the duration of the actions and this is not in general possible.

7. Example

In this section we present an example of a decision-tree (figure 5). The user will not need to write the decision-tree in this form; instead there will be a user-interface that will support the construction of the tree. The example is mostly constructed with the aim of showing the capability of the system and does not claim to be correct in terms of military tactics. Parts of this decision-tree have already been used in previous sections. Here we present a few brief comments:

- **(do-fire x) turn** is a sequential action. **turn** and **release-decoy** together constitute a concurrent action;
- if the condition *(selected-enemy x)* is true then *(selected-enemy x) → 6* adds 6 to the priority of the action at the end of the branch;
- if an action has priority less than or equal to 0, it is not performed;
- *(> (time-since-fired-at x) 5)* means that the agent that is making the decisions has fired at the aircraft *x* more than 5 time units before or has never fired at it;
- The condition *(< (distance-to x) 100)* means that the agent that is making the decisions is at a

distance less than 100 spatial units from the agent x;

- The condition *selected-for-wing* is true if an enemy has already been selected for the wing-man;
- (*set-intercept x*) records the decision of intercepting x. (*intercept x*) is true if x is the enemy that the agent has previously decided to intercept;
- (*go-toward x*) makes the agent go toward the last recorded position of x;
- (*send-command agent command*) means that the agent that receives the command updates the state. The value can then be tested in the context of the updated state and may influence the behavior of the agent. For example (*send-command wingman (selection-enemy x)*) will make the wingman add to the state that the selected-enemy is x.

We have tested this decision-tree in a scenario with two opposing sections. Let us consider step by step some of the decisions that the leader of one of the sections makes. First, the action *go-to-mission-points* is performed and the agent starts moving toward the first of these points. When the first aircraft of the enemy section is detected, the agent selects this enemy as primary target, (*selection-enemy x*), and decides to intercept it, (*set-intercept x*). In the next cycle the interception is started (*go-toward x*). When the second enemy is detected a command is sent to the wingman to select this enemy as primary target and it is recorded that the wingman has now a selected enemy. When one of the enemies is close enough, a missile is fired. However in the meantime one of the enemies has also fired a missile and our agent receives a missile warning. So it turns and it releases a decoy. When the turning action is completed, the speed is increased. The scenario continues then to evolve with several interceptions, firings and avoiding of missiles until just aircraft of the same side are left.

8. Conclusions and future work

In this paper, we have presented preliminary results of a project on the design and implementation of intelligent agents for air combat simulation. In particular, we want to permit air combat experts to implement such agents without the aid of computer expertise. In order to achieve this, we are attempting to find a good balance between simplicity and expressiveness in the means given to the user to specify the behavior of the agent. We intend to continue the development of the system and to

evaluate it both with respect to performance and user issues.

Our approach has the advantage of flexibility in responding to situations, as the priority of the actions is changed dynamically depending on the actual situation. At the same time describing complex situations is made easier by the fact that decisions are taken at several levels, from general decisions to specific ones, and also by the fact that the decisions do not need to be exclusive. In fact, several branches of the decision-tree can be visited at the same time and several alternative actions can be taken into consideration. Dynamically changing priorities establish which actions are the ones that are performed. Decisions are reconsidered at each step of the simulation and this allows a quick reaction to changes in the situation. At the same time the agent can also follow strategies performing sequences of actions.

In this paper we have mainly considered the decision mechanism in itself. In the future we also intend to consider issues such as communication, coordination and tracking of enemies. Further it seems important to implement a more sophisticated handling of the interruption of sequential actions which, for example, would make it possible to continue an interrupted sequence of actions. It would also be of interest to consider a change in the priorities of the actions due to an assessment of the resulting situation after performing the action.

9. Acknowledgment

We would like to thank M. Tambe, G. Frisk, Johan Görsjö, Jenny Andersson, and Patrick Lambrix for their helpful comments. This work is partly supported by the Center for Industrial Information Technology, Linköping University (CENIIT), and the Swedish National Board for Industrial and Technical Development (NUTEK).

10. References

- Decision-Science Application Inc. (1991) "The TAC BRAWLER air combat simulation management summary", *Technical Report 907*, DSA, Virginia, Usa.
- Laird, J. E., Jones, R. M., and Nielsen, P. E. (1994) "Coordinated Behavior of Computer Generated Forces in TacAir-Soar", in *Proceedings of the Fourth Conference on Computer Forces and Behavioral Representation*, Orlando, FL.
- Saab Military Aircraft (1995) "User Guide TACSI", *Technical Report TUCU-MI- 95:103*, Saab Military Aircraft, edition 3.1.
- Tambe, M., Johnson, W. L., Jones, R. M., Koss, F., Laird, J. E., Rosenbloom, P. S., and

Schwamb, K. (1995a). "Intelligent Agents for Interactive Simulation Environments", *AI Magazine* 16(1).

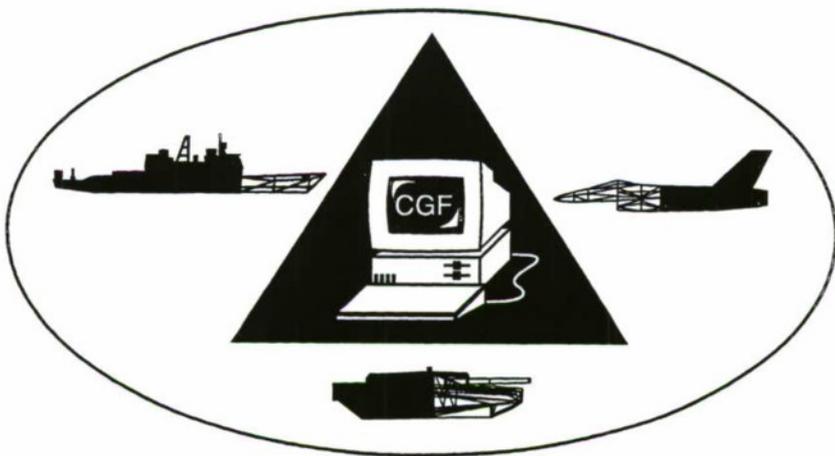
Tambe, M., and Rosenbloom, P. S. (1995b). "Resc: An Approach for Dynamic Real-time Agent Tracking", in *Proc. IJCAI'95*.

11. Authors' Biographies

Silvia Coradeschi is currently doing research at the Department of Computer and Information Science at Linköping University. Her interests are in the areas of artificial intelligence and computer generated forces.

Lars Karlsson is a graduate student at the Department of Computer and Information Science at Linköping University. His interests are in the area of artificial intelligence, in particular planning and autonomous agents.

Anders Törne is associate professor at Linköping University and director of the real-time system laboratory. His interests are in the area of specification, design, and verification of complex real-time systems.



The Automated Wingman: An Intelligent Entity For Distributed Virtual Environments

Capt. Sheila B. Banks, Ph.D., Professor Eugene Santos, Ph.D., Lt. Col. Martin R. Stytz, Ph.D.
Virtual Environments, 3D Medical Imaging, and Computer Graphics Laboratory
Artificial Intelligence Laboratory
Department of Electrical and Computer Engineering
Air Force Institute of Technology
Wright-Patterson AFB, OH 45433
sbanks@afit.af.mil, esantos@afit.af.mil, mstytz@afit.af.mil

1. Abstract

Research in Computer-Generated Forces (CGFs) has developed the ability to create a variety of human and computer-controlled entities that can operate within a distributed virtual battlespace. Unfortunately, it is often easy to distinguish between human-controlled and computer-controlled entities because of the predictable nature with which the computer-controlled entities behave. This defect may allow training subjects to identify the CGFs, predict their behavior, and defeat them using methods that would fail against human-controlled entities. Consequently, we undertook the Automated Wingman research project. The Automated Wingman is implementing a computer generated aircraft system that exhibits correct human behaviors without mimicking the human reasoning process by relying upon fuzzy logic as its primary reasoning mechanism. In this paper, we present the current state of the Automated Wingman's development as a realistically behaving computer generated aircraft system. In our view, in light of continually developing requirements, the knowledge base and system architectures are keystones to the success of the research. Therefore, we discuss the system architecture and knowledge architecture methods we use to maintain independent system components and to enable rapid evolutionary and exploratory prototyping for both of these aspects of the Automated Wingman. We conclude with requirements for future work on this project.

2. Introduction

Within the modern battlespace, the air component is a decisive factor in determining the outcome of an engagement. To date, however, there has been under-representation of aircraft entities within Distributed Interactive Simulation (DIS)-based distributed virtual environments (DVE). This is partially due to the expense of developing aircraft simulators that are DIS-compliant and partially due to the difficulty in developing aircraft computer generated forces (CGFs). To address this problem, we are developing a CGF that can be used to realistically increase the number of aircraft within the DVE while minimizing the cost of achieving a higher aircraft entity count. Our research

is aimed at developing aircraft CGFs that exhibit the complex characteristics of human decision making and behavior. Rather than initially attempting to develop an entity with a complete set of pilot tactical skills, we chose to begin with a simpler problem, that of modeling a wingman's behavior. This project is called the Automated Wingman. During operation, the Automated Wingman flies at the wingman's station in support of a lead, manned simulator but with enough intelligence to be indistinguishable from human controlled entities.

Because of the requirement for intelligent behavior apart from the lead and the need to deal with uncertainty, ambiguity, and approximation to model human behavior, we chose to use fuzzy logic as the basis of the AW's decision making capabilities. Fuzzy logic is an artificial intelligence technique that enables the entity to mimic human behaviors by dealing with ambiguity and uncertainty in a way that traditional logic cannot. The Automated Wingman uses a fuzzy expert system to select a tactical maneuver or set of maneuvers and control their execution. To use this approach, we require appropriate knowledge bases and linguistic variables, and the production rules to manipulate them. The Automated Wingman fuzzy expert system uses a hierarchy of knowledge bases for decision making and knowledge storage. The fuzzy expert system provides the AW with a reasoning capability while the knowledge bases provide the information required to select appropriate tactics, determine the required maneuvers to implement those tactics, and fly the maneuvers.

Current work on the Automated Wingman requires that we concurrently develop several AW capabilities. The knowledge engineering tasks for this year are to develop the knowledge bases defining tactical situations that must be processed by the AW and to refine the fuzzy logic-based decision-making capability (including fuzzy sets, fuzzy variables, and fuzzy variable hierarchy) for assessing tactical situations. We are also extending the knowledge bases to support 1) two airframes operating within four mission types, 2) complex inter-entity behaviors for cooperative formation flying between multiple AWs, and 3) an improved on-board planning

capability. A final important requirement is development of a capability to model pilot skill levels ranging from novice pilot to the expert pilot level. This final capability affects all the knowledge bases in the AW. Because we are attempting to address these requirements simultaneously, we require a methodology to guide us in managing both the knowledge base development process and the AW software architecture.

This developmental approach places severe strain on the software architecture and the structure of the knowledge base. The strain arises from the need to accommodate changing implementation and performance requirements as well as continual improvement in the AW's reasoning capability. To accommodate the instability of requirements and the accelerating change of pace in the underlying technologies, we use a software architecture suitable for implementing and maintaining applications developed using a modified rapid prototyping approach. A key aspect of this architecture is the Common Object Database (CODB) (Stytz, et. al., 1996). To accommodate the need for continual improvement in the AW knowledge base, we use the Rapid Exploratory and Evolutionary Prototyping (REEP) methodology.

The next section presents background information concerning the Automated Wingman project. Section Three presents a description of the Automated Wingman's software architecture. Section Four presents our rapid prototyping approach to developing and refining the AW's knowledge bases. In section Five we describe the current status of the Automated Wingman and Section Six contains expectations for future work.

3. Background

The Automated Wingman seeks to improve the state of the art for CGFs by using fuzzy logic as its core reasoning mechanism. Because its use is central to the capabilities of the Automated Wingman and to our approach to developing its knowledge bases, in this section we will present a brief introduction to fuzzy logic and provide an example of its use.

Fuzzy logic provides the Automated Wingman with a means to represent and reason with uncertain data, ambiguous terms, and approximations. Therefore, a fuzzy logic system is potentially capable of dealing with situations that cause difficulty for systems that use traditional Boolean logic. Because humans continuously deal with uncertainty, ambiguity, and approximation, we believe that any CGF required to exhibit human behaviors must also be able to deal with uncertainty, ambiguity and approximation. Fuzzy logic provides this ability.

The strength of fuzzy logic is its evasion of Aristotle's Law of the Excluded Middle (Kosko). Aristotle stated in his "Laws of Thought" that an assertion can be either true or false, but not both. Therefore, the middle (partially true and partially false) is excluded. The Law of the Excluded Middle is a central concept behind traditional logic systems, such as Boolean logic. However, there are many commonplace situations where traditional logic fails. These situations are called paradoxes. Although paradoxes are often dismissed as trivial and meaningless by mathematicians, these paradoxes lie at the core of the real world problems faced by computer scientists and expert system designers who have to contend with the lack of expressability of traditional logic systems.

In a fuzzy-logic system of reasoning, an assertion may have a degree of both truth and falseness. While this may seem contradictory, it is a common way of representing situations. For example, consider a piece of teal matte board and the assertions "the board is blue" and "the board is green". Depending upon the shading, we may say that the assertion that the board is blue is true to degree 0.6 (out of 1) and the assertion that the board is green is true to degree 0.4. We now have two assertions, both of which are true to a degree. We can reason with these assertions by factoring in the degree of truth of each assertion to arrive at a conclusion that considers all of the available information.

The concept within fuzzy logic relied upon by the Automated Wingman is that of a linguistic variable. A linguistic variable, such as temperature, describes a quantity or an idea that is best represented by fuzzy sets, called term sets. For example, fuzzy sets for temperature could be hot, warm, and cold. The value of the linguistic variable can be assigned to one of these term sets. For example, if we agree that 100° C is hot, then we can say that the temperature of boiling water is hot. A more powerful technique is to "fuzzify" a crisp value and determine the term set(s) to which the crisp value belongs, allowing the linguistic variable to be evaluated as the union of its fuzzy sets. The linguistic variable then takes on the value of all the term sets that apply, not the crisp value itself (Zadeh, Schwartz). To better illustrate the use of fuzzy logic within the Automated Wingman, we will discuss it within the context of the knowledge base design for its flight control system.

The flight control knowledge base must provide the AW with three independent axis of control. These are *altitude*, *heading*, and *thrust*. Using the linguistic variables in Table 1, we developed production rule graphs for each of these axes of control. These rule graphs show how the linguistic variables combine to describe the state of the

Automated Wingman along each axis and the correct action that the Wingman should take in response to the state. As a result, the Automated Wingman can control its own airplane entity.

Table 1: Linguistic Variables in the Automated Wingman

LINGUISTIC VARIABLES
Current Relative Altitude
Projected Relative Altitude
Vertical Velocity
Vertical Velocity Difference
Desired Vertical Velocity Difference
Projected Vertical Velocity Difference
Vertical Acceleration
Total Acceleration
Current Relative Airspeed
Projected Relative Airspeed
Projected Airspeed Difference
Relative Heading
Range
Lead Bearing
Bank Angle

Although Flight Control consists of three axes, space limitations permit us to describe only the Altitude axis. Altitude is assessed using the ClimbRate linguistic variable and term sets. The ClimbRate variable assesses the climb rate of the aircraft that is required to achieve the desired altitude. This variable considers the current altitude relative to the desired altitude and the altitude difference at some time in the

future given the current velocities and accelerations. This linguistic variable relies, in turn, on other linguistic variables to determine the appropriate value.

Figure 1 presents the Climb Rate Rule Graph for the Automated Wingman. Each of the bubbles in the decision tree represents a term set that describes a quantity relevant to climb rate. The Flight Controls module determines values for all these term sets and then navigates this graph to determine the value for ClimbRate. For example, at the top of the graph the current value of AttachMode is checked. If AttachMode is "Attached" then the term set "CurrentRelativeAltitude" is examined. If that is "Nil" then the VerticalVelocityDifference variable is checked. If that is also "Nil," then the ProjectedRelativeAltitude variable is tested. A "Nil" for that variable indicates that the Wingman should maintain the current vertical velocity (climb rate). However, because the term sets are fuzzy, the other paths through the graph may also apply but with less weight. To determine which term sets apply, each path through a term set evaluated as greater than 0.0 is examined. The other paths' weights will "spread" the value of ClimbRate out so that it can encompass all five term sets, Decrease, Dip, Maintain, Bump, and Increase, to a degree. In general, all the term sets apply to some degree in every situation but most will apply to a near-zero degree. The variability in the degree with which each term set applies to the linguistic variable is the foundation for the power of this form of reasoning.

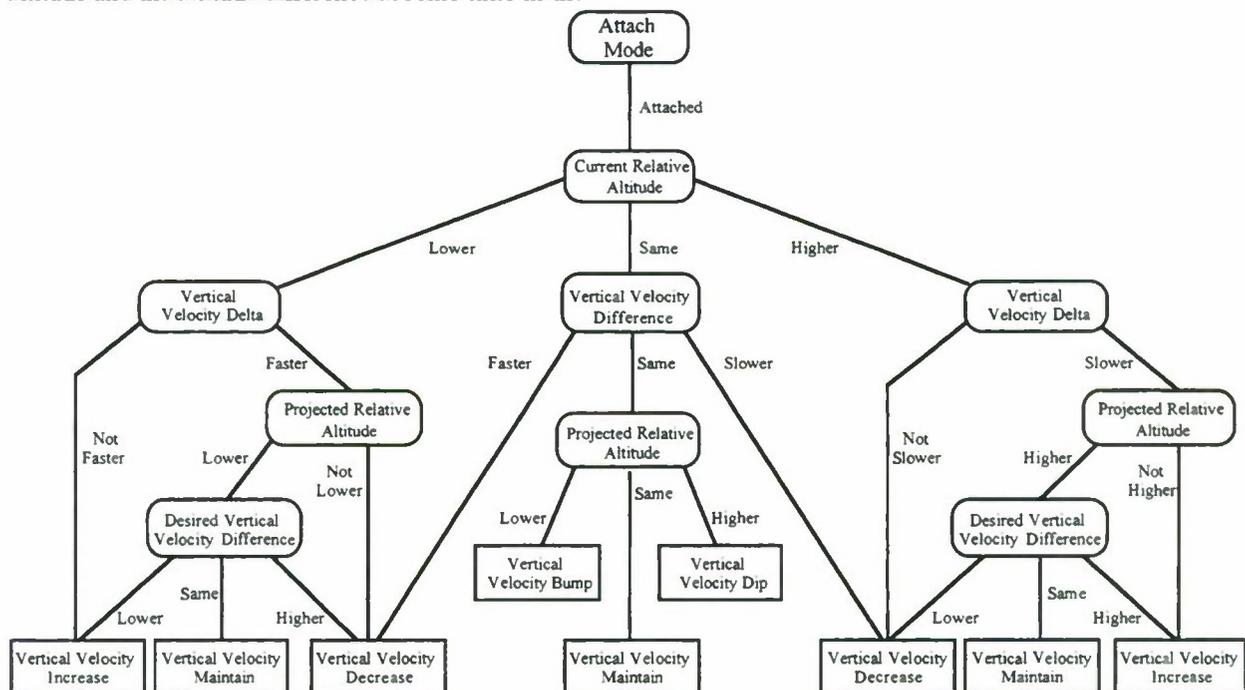


Figure 1. Climb Rate Rule Graph for the Automated Wingman

The Automated Wingman is not the first attempt at creating a realistic CGF that exhibits human behaviors. Several others have tried with varying degrees of success such as Tac-Air Soar (Laird, et.al. and Tambe, et.al.). Tac-Air Soar builds upon the Soar architecture for general intelligence and reasoning. Tac-Air Soar is the most successful of the current aircraft CGFs and it has participated in several exercises, including the STOW-E (Europe) exercise. During STOW-E the Tac-Air Soar team was able to field aircraft entities, conduct independent force type missions, and fight against manned simulators in a limited fashion. Unlike the Automated Wingman, Tac-Air Soar does not handle uncertainty in its decision making process.

There are several requirements driving our AW design decisions. These are DIS compatibility, autonomous flight control, automatic route planning, within visual range combat, beyond visual range combat, and route planning. The Automated Wingman must be DIS compliant to perform its mission as a CGF. Like any pilot, the Automated Wingman must be able to fly its own plane, know where it is going, and know how to get there using maneuvers available to human pilots. The AW must also, at times, operate as an intelligent entity independent from its lead aircraft simulator. Therefore, the AW must have the ability to independently plan within the context of a mission plan and commands from the lead aircraft. The requirements for autonomous route planning and flight control support these capabilities. The Automated Wingman must also select a suitable tactic or maneuver based on the current situation in light of its mission and near term goals. Assessment of the current situation requires the ability to orient sensors in the appropriate direction, then fuse and interpret the incoming raw sensor data. The interpretation and assessment of incoming sensor data forms the basis of situational awareness for a CGF. Using its knowledge about the current world situation, along with knowledge concerning tactics and doctrine, weapon employment capabilities, and voice commands from the lead pilot, the Automated Wingman then selects an appropriate pilot behavior that will be indistinguishable from that of a human controlled entity. Satisfying these requirements led to the development of the software architecture presented in the next section.

4. Automated Wingman Software Architecture

Our motivation for the development of a general architecture to support the AW was to provide a basis for the design of broad classes of CGFs. While each class of CGF has its own unique characteristics and performance requirements, we contend that there are many factors common to all classes and that these can

be successfully reused across all classes of CGFs. We have noted that simply crafting CGFs primarily from the viewpoint of emphasizing differences often results in only a few highly specialized types of CGFs. This is typically due to the amount of knowledge engineering necessary to effect intelligence and the focus on guaranteeing the unique behaviors of the CGF. Without a general approach to constructing CGFs, it is likely that little or no information will be transferable from class to class, or even between entity types in the same class of CGF.

Aside from the Soar projects, little work has been done to address the following issues of CGF construction: 1) approximation of human behaviors, 2) computational efficiency, 3) ease of knowledge-engineering, and 4) scaleable performance. Our goal is to provide a general architecture for CGFs which naturally accounts for "variety" in a given type of CGF as well as detail a general approach for organizing and building vastly different CGFs such as tanks versus aircraft. Given the continuous changing nature of CGF requirements as we learn more about them, an evolutionary and exploratory approach to knowledge engineering, such as the REEP methodology, discussed in the next section, is also required. Our architecture consists of highly modular components where interdependencies are well-defined and minimized.

The architecture we developed is based on several precepts. The first is that future architectures should focus on reducing programmer costs, even at the possible expense of marginal processing inefficiencies. We believe that this tradeoff is wise because the growth in CPU power that will occur over the period of system development will offset the minor processing inefficiency costs that are introduced by minimizing programming costs. Note, however, that this strategy only allows for marginal inefficiencies. We contend that code within an object should be clean and tight, however we believe that inter-object communication should be open and clear with a minimum of coupling.

The second precept is that the development cycle for the Automated Wingman, as in most research and development projects, will include a series of revisions to the requirements and additions to its desired capabilities because the basic system requirements continue to evolve. These changes range from changing protocol data unit (PDU) formats to introduction of new behaviors, such as infrared sensor management, close-air support, or smart bombs delivery, into the system. As a result, a formal requirements analysis process is not a worthwhile undertaking because end-users will generally not be aware of defects and shortcomings in the system until the system is in operation and operational tests reveal new requirements. As a

result, the architecture should be developed to support both exploratory and evolutionary rapid prototyping in order to reveal new requirements and to test solutions.

A third precept is that the push to attain improved performance and the strain of meeting delivery deadlines increases the entropy of any design, until the design concept becomes blurred. The most obvious symptoms of this occurrence are the use of global variables, global functions, and the disappearance of private data items. The architecture should, therefore, address the problem of increasing entropy by erecting entropy firebreaks between the major objects in the system and mechanisms that encourage the programmer to remain within the architecture rather than circumvent it. As a result, the architecture relies upon an object-oriented design of its major system components, containerization, and a common object database to manage public data.

A fourth precept is that the components of the airframe (aerodynamics model, avionics systems, and weapons packages) should be rapidly modifiable. Therefore, these components should be realized as separate objects that have a clean, robust interface to the remainder of the system. In addition, the airframe components should be built upon validated models. The reasoning components that use the outputs from the airframe CGF components should be separate.

A final precept is that expanding system requirements will cause the knowledge base and reasoning system to be modified and adapted to new requirements throughout the life of the project and the subsequent fielded system. For example, impending requirements for CGFs are the capability for multiple skill levels within the CGF, a capability for the CGF to direct its attention to specific environment components, a capability to control smart weapons, a capability to change its reasoning pattern to adapt to new avionics capabilities, and a capability for sensor management. As a result, we concluded that these knowledge and reasoning components should be structured so that the knowledge base and reasoning system are separate. Additionally, the analysis and action components of the reasoning system should be separate components as well. Furthermore, since we are using fuzzy logic, we should implement each of these components as a hierarchy of objects that serve to aggregate information and dynamically limit the search space.

Our system architecture, see Figure 2, used these five precepts to guide the architectural definition. Within the architecture we use containers, which are data

structures used to move large amounts of structured data between system components, to manage and control inter-component communication. The main AW components are specified as objects. These objects are the Pilot Skills Component (PSC), the Active Decisions Component (ADC), the Physical Dynamics Component (PDC), the Common Object Database (CODB), the World State Manager (WSM), and the Environment Database. Each of these objects are, in turn, hierarchically defined as a set of objects that use the containers to communicate with the other components of the AW via the Common Object Database. The CODB holds all public data for the AW and all system components may access the CODB for data from other system components. The World State Manager is responsible for maintaining a complete description of the state of distributed virtual environment as communicated using DIS-formatted PDUs. The Pilot Skills Component, the Active Decisions Component, and the Physical Dynamics Component are discussed in detail later in this section. In our system, component development is accomplished using a rapid prototyping approach that uses both exploratory and evolutionary prototyping to extract system requirements and refine requirements solutions.

Figure 2 also presents the basic CGF architecture and its relationship to the other system components. In its most abstract form, a CGF consists of three components: a PDC, a PSC, and an ADC. The PDC encapsulates all the physical attributes and properties of the CGF. For example, in the AW, this component includes the aerodynamics model, entity-specific properties, aircraft capabilities, weapons load, sensors, damage assessment, and physical status. In addition, the PDC contains the processes for computing physical state changes such as updating object position in the virtual environment. The PSC consists of those portions of the CGF that need to vary between individual entities within a type and class. This component serves to model the skills and ability of the pilot of the entity. For an aircraft entity, the components of the PSC consist of the pilot's ability to maintain situation awareness and to execute tactical and flight skills. These PSC components play an integral part in the decision making ability within the ADC. The ADC encompasses the intelligent decision making processes and the knowledge necessary to properly drive them. This includes the overall mission, goals and objectives, plan generation, reaction time, and crisis management ability, etc. Clearly, the ADC must accomplish many of its activities in real-time.

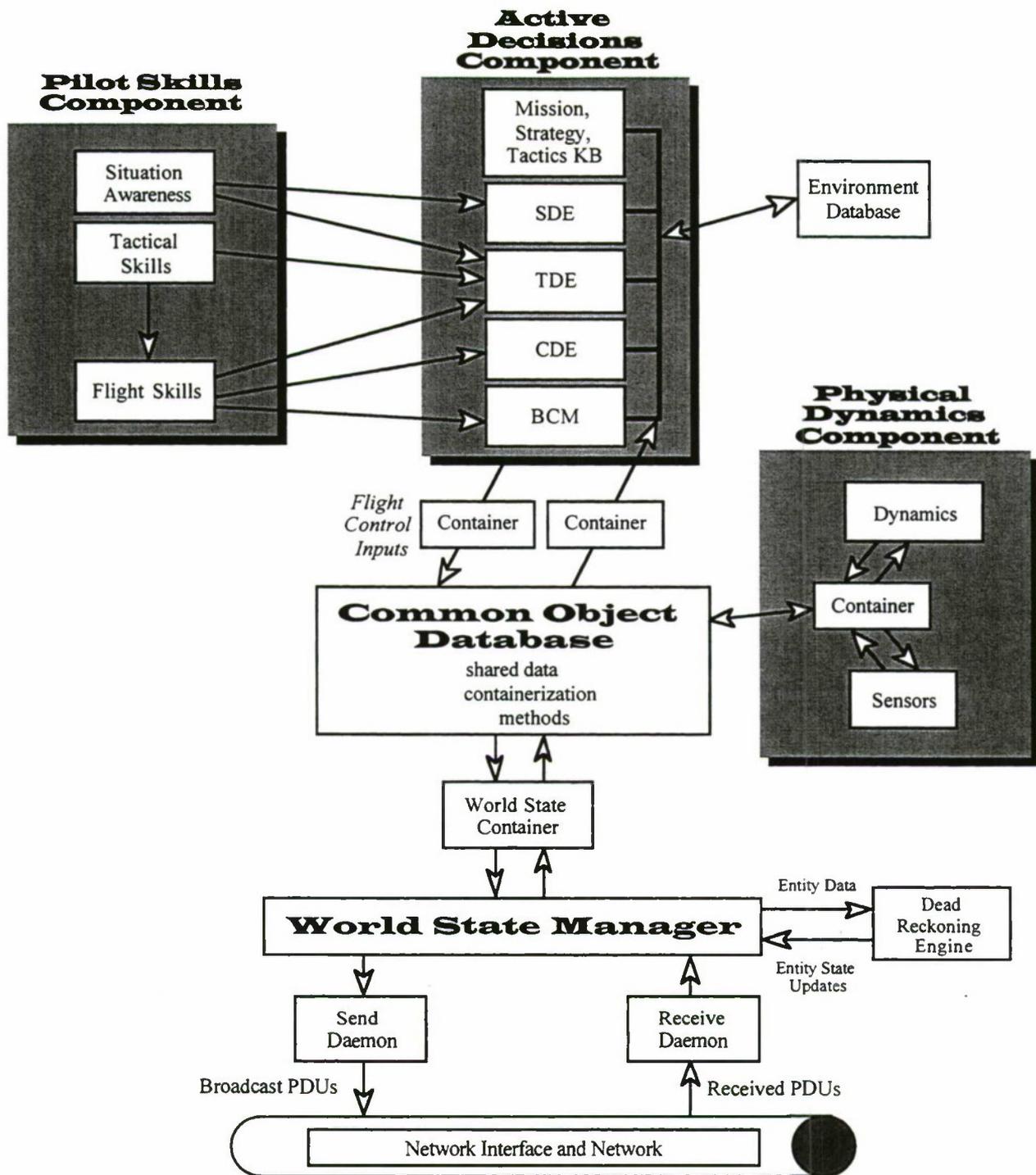


Figure 2. Automated Wingman System Architecture Incorporating the Common Object Database

We use this tri-fold separation of the components of the CGF in order to insure that changes are isolated and do not propagate throughout the system. The PDC is only responsible for the basic entity maneuver information, and operates completely unaware of the status of the other system components. Likewise, the ADC is solely responsible for decision making and only knows about the physical

component's status based upon the data placed in the CODB. The PSC is more closely tied to the ADC than the PDC because the ADC is responsible for computing control outputs for the entity based upon the modeled pilot's skills. The PSC supplies a description of the pilot's ability to the decision making component so that the decision can be appropriately constrained by the pilot's abilities.

The division of capabilities between these basic components lessens the system level impact of any requirements changes in the PDC, PSC, or ADC.

The PSC and PDC contain all the information and status required to portray an aircraft model and model its pilot's ability. The PDC encapsulates the entity state information and the PSC contains a representation for all the pilot skill variables. The key aspect of these two components is that these subsystems are completely parameterizable, and hence rapidly reconfigured and reused. We isolate entity control skills into the PSC because this separates the ability to parameterize the operator's capabilities from the decision making mechanisms used by the operator. Through this parameterization, any number of CGFs of a given type may be generated using a given ADC so that each entity has its own unique set of operator skills. The PSC models the pilot's skills as a hierarchy of capabilities. The lowest level of the hierarchy contains the atomic skills for the pilot, such as ability to perform a bank, highest sustainable g-force level, ability to acquire a target, ability to operate a weapon system, etc. Subsequent levels of the hierarchy are web-like interconnections between these skills. This scheme allows us to compose more complex skills from elementary skills and to compose the higher level skills using a careful weighting of the appropriate elementary skills. The drawback to this approach is that the atomic skills must be carefully chosen and crafted so that high level skills have the desired performance. The PDC and PSC do not, however, perform decision making based upon the information they store. The decision making task is solely the responsibility of the ADC.

Decision making in the ADC is not based on a traditional goal-driven planning approach. Instead, the ADC contains the fuzzy goal-planner that allows certain subgoals to remain unsatisfied but still have the supergoal satisfied. This decision making flexibility permits for a much wider variety of possible behaviors and provides additional decision-making elasticity to allow the CGF to achieve its mission in the face of uncertainty. That is, the system can tolerate uncertain satisfaction of subgoals and then use it as a measure. Also, the fuzzy approach provides a method for optimization when various subgoals are applicable but only one is desired. This use of fuzzy logic adds another behavioral distinction that can be exploited to create a diverse mix of entities.

The ADC is the heart of the Automated Wingman, and holds the fuzzy logic decision engines. There are four primary reasoning modules of interest: the strategic decision engine (SDE), the tactical decision engine (TDE), the critical decision engine (CDE), and the basic control module (BCM). The ADC also contains relevant knowledge-bases specific to these

reasoning modules. The SDE handles strategic matters related to accomplishing mission goals by continuously re-evaluating the completion status of mission objectives and re-planning to achieve the objectives in a deliberative fashion. To execute its plans, the SDE then requests the TDE to carry out the near term (tactical) objectives. The TDE operates under the direction of the SDE to manage near-term situations and determine a fine-grain course of action for imminent tactical situations. It then implements those actions as requests to the BCM. For example, for an aircraft, the TDE transmits stick and throttle settings to the BCM. The TDE is less deliberative than the SDE and must perform its functions in real-time. The CDE is a purely reactive reasoning system that deals with critical situations the AW might encounter. Its purpose is to enable the AW to survive a life-threatening situation, and it operates independently of mission goals and objectives. For example, pilots are trained to respond in a certain fashion when presented with a threat such as an approaching surface-to-air missile. To operate effectively, the CDE monitors the world state (in the container passed from the CODB) passively until a critical situation is detected. The CDE then assumes control of the AW until the crisis has passed. During the crisis, the SDE and TDE monitor the AW's state so that they may resume control after the crisis has passed. Lastly, the BCM processes the requests of the TDE and CDE to pass as flight control inputs for the AW. Processing the requests takes into account the state of the PDC and PSC most relevant to the requests. For example, the BCM filters its flight control decision outputs using parameterized pilot ability ratings to execute a maneuver before it is applied to the aircraft's control inputs. The ADC could initially operate with only the BCM.

The above decomposition of the ADC maintains component independence. Furthermore, knowledge-base decomposition mirrors that of the decision engines, allowing the various knowledge-bases to be constructed and tested independently. By modularizing our decision engines and the knowledge-bases in this fashion, traceability and validation of the CGF behaviors can be much more easily achieved than previous approaches to knowledge engineering CGFs. This improves our prospects for correctly revising and identifying new behaviors during our development life-cycle. Finally, within each sub-module or knowledge-base, additional hierarchies can be imposed to further increase the possibility of re-use when constructing other similar CGFs.

5. Rapid Evolutionary Prototyping Of The Knowledge Architecture

To facilitate progress in our research projects, we use a modified rapid evolutionary exploratory prototyping (REEP) approach in conjunction with exploratory prototyping for developing and improving the AW knowledge architecture and implementation. Rapid evolutionary prototyping is the use of prototyping techniques to achieve incremental improvements in knowledge base implementation and design. The approach arose from the realization that users need an operational system to measure against their

expectations and needs. By combining exploratory and rapid evolutionary prototyping, users can use functional systems to uncover and validate requirements and facilitate implementation solutions. Just as important, these techniques allow us to make progress on the system in the face of continually uncovered requirements and simultaneous modification of multiple system components. The combination of exploratory and rapid evolutionary prototyping also allows us to manage the prototyping process and to incorporate prototyping results into the AW.

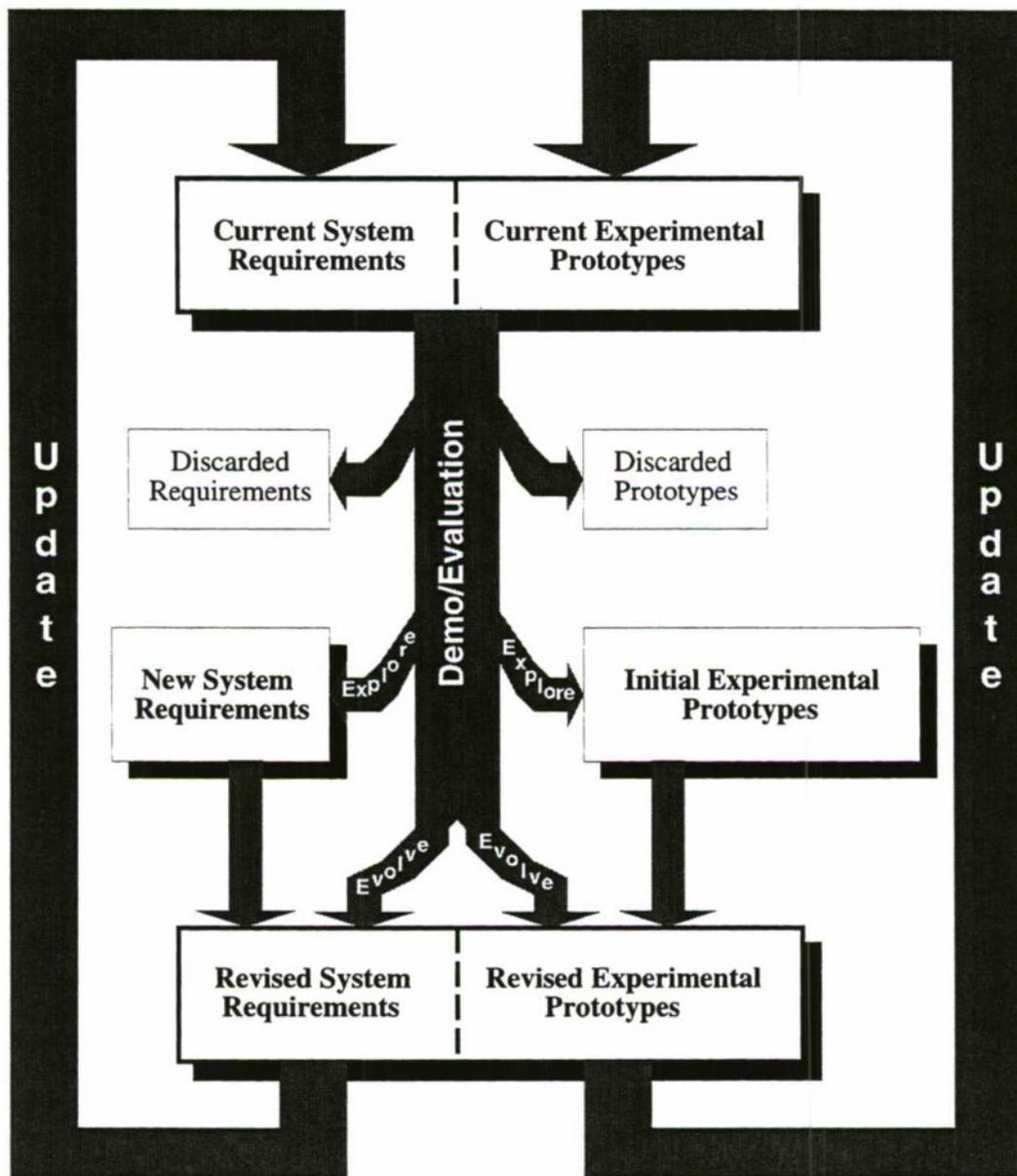


Figure 3: Process Flow in System Development.

Figure 3 shows the general process flow from exploratory prototype to evolutionary prototype. To provide a starting point for knowledge base development, we construct a system that satisfies the known baseline requirements at the beginning of the project. Using feedback from users in our laboratory and demonstrations of the systems, we determine requirements that were unknown when we established the baseline requirements and refine and expand upon the requirements that the baseline addressed. As a result of this continual feedback and experimentation process, we arrive at a revised set of requirements, design solutions to the requirements, and implement them.

When we discover new requirements, we use exploratory, or partial, prototypes to examine specific means for addressing the requirements because complete prototypes are expensive to build. Exploratory prototyping is the use of a prototype to refine a requirements definition or to examine an implementation solution within the context of an operational system. We do not require that the exploratory prototype be fully functional. The prototype may be retained in the system for further development or discarded. If we discard the prototype, then subsequent evolutionary systems incorporate the lessons learned from the exploratory system. We learned that the exploratory prototyping approach is valuable when the main question to be answered is implementation related and there is little experience in building the desired software solution. We have used exploratory prototypes to address both small and large requirements, thereby allowing us to implement and assess potential solutions in a few days or weeks.

Because we use rapid evolutionary prototyping and object-oriented programming, we reuse software components of the baseline evolutionary architectures whenever possible. Our intent is that successive revisions to the design do not require major reworking of the software components or of the software architecture, although this certainly can happen. The evolutionary systems themselves evolve over time, but are always complete systems. Each new system tends to bring to light new requirements. These new requirements determine the experiments we perform with the next set of exploratory prototypes. We then incorporate the solutions developed via the exploratory prototypes into the next evolutionary prototype. The solutions typically take the form of new objects or revisions to existing objects.

This combination of prototyping techniques allows us to manage the process of defining and refining requirements and implementation approaches within components of the system even as system

requirements, design, and implementation evolve and develop.

6. Current Status

At the heart of the Automated Wingman is a fuzzy expert system that uses a hierarchy of knowledge bases for decision making and knowledge storage. The fuzzy expert system provides the Automated Wingman with a reasoning capability while the knowledge bases provide the information required to select appropriate tactics, to determine the required maneuvers to implement those tactics, and to fly the maneuvers.

The PDC has completed the modifications necessary to incorporate a new, accurate aerodynamics model. With this model in place we are able to rapidly change the dynamics behavior of the aircraft CGF by simply changing a parameter file. We use this capability to implement a variety of aircraft entities. The PSC is under development at this time. The key aspects of this development project are continuation of an assessment of the key components of pilot skills and assessment of the knowledge domain.

The ADC is the main focus of our current work. All four components within the ADC are in begin developed simultaneously; however, the TDE and BCM are key to our progress this year. Current development efforts for the TDE include incorporation of defensive counter air and close air support tactical decision making capabilities. The current BCM flight control capability exhibits improved capabilities over last year's prototype (Edwards, 1996).

The current implementation of the Automated Wingman operates on a Silicon Graphics workstation. The Automated Wingman achieves an update rate of approximately 15 cycles per second. All application software, except for Fuzzy CLIPS, is written in C++.

To date, the Automated Wingman project has demonstrated the viability and feasibility of a fuzzy logic based CGF. We have a fundamental design that is flexible and ready to serve as the foundation of future efforts on this project. Our implementation has shown that a hierarchy of fuzzy linguistic variables can be used to control a dynamic process in an airplane. However, the AW is far from complete. The next section presents our ideas as to future development.

7. Future Work

Further developmental work will address the need to incorporate the Command and Control Simulation Interface Language (CCSIL) to provide a means for

the lead aircraft to verbally issue commands to the AW. The SDE strategic planning capability must be further enhanced and knowledge added to allow the AW to transition to autonomous operation. The PSC knowledge bases must be expanded to include multiple pilot skill levels that degrade under pilot stress and fatigue factors.

Necessary future work must also address developing a sensor fusion and sensor management capability using fuzzy logic within the AW. The AW sensor package needs to be augmented and the pilot skills should be modified so that the AW exhibits different behaviors and capabilities based on time of day and weather conditions. The AW will be given an updated weapons store and the ability to choose these weapons based upon different situations that can occur during a mission. Therefore, an updated weapons selection knowledge base is required.

On a broader scale, a methodology to take a model of a subject and create a computer generated entity that exhibits the behaviors of that subject is needed. On a fundamental level, this requires a formal design technique for the linguistic variables and term sets to be used by the automated entity. The current state-of-the-art for their design is merely trial and error, which is both time consuming and error prone. A set of techniques that would identify the of linguistic variables and their definitions would dramatically reduce the time required to implement and validate the system. Finally, a rigorous test methodology for validation of linguistic variables that ties required behavior to the associated variables and term sets is needed. Using our progress on the Automated Wingman as a guide, we have begun to address these issues.

8. References

- CLIPS Users Guide*, (1993) National Aeronautics and Space Administration, Cape Canaveral, FL.
- Edwards, M. and Stytz, M. R. (1996) "The Fuzzy Wingman: An Intelligent Companion for DIS-Compatible Flight Simulators", *The SPIE/SCS Joint 1996 SMC Simulation Multiconference: 1996 Military, Government, & Aerospace Simulation Conference*, vol. 28, no. 3, New Orleans, Louisiana, 77 - 82.
- Giarratano, J. and Riley, G. (1994) *Expert Systems: Principles and Programming*, PWS Kent, Boston.
- Knowledge Systems Laboratory. (1994) *Fuzzy CLIPS Version 6.02A User's Guide*, Institute for Information Technology, National Research Council Canada, Ottawa, Ontario, Canada.
- Kosko, B. (1993) *Fuzzy Thinking: The New Science of Fuzzy Logic*, Hyperion, New York, NY.
- Laird, J. E., et. al.. (May 1995) "Simulated Intelligent Forces for Air: The Soar/IFOR Project 1995," Proceedings of the Fifth Conference on Computer Generated Forces and Behavioral Representation, Orlando, FL.
- Schwartz, D. G. (1991) "A System for Reasoning with Imprecise Linguistic Information," *International Journal of Approximate Reasoning*, Vol 8, 463-468.
- Stytz, M. R., Adams, T., Garcia, B., Sheasby, S. M., and Zurita, B. (1996) "Developments in Rapid Prototyping and Software Architecture for Distributed Virtual Environments," *IEEE Software*, vol. 12, to appear.
- Tambe, M., Johnson, W. L., Jones, R. M., Koss, F., Laird, J. E., Rosenbloom, P.S., and Schwamb, K. (Spring 1995) "Intelligent Agents for Interactive Simulation Environments," *AI Magazine*, vol. 16, no. 1, 15 - 40.
- Zadeh, L. A. (1975) "The Concept of a Linguistic Variable and its Application to Approximate Reasoning," *Information Sciences*, Vol 8, 199-249 and 301-357.

9. Author's Biographies

Sheila B. Banks is an active duty Captain in the U.S. Air Force serving as an Assistant Professor of Computer Engineering at the Air Force Institute of Technology, Department of Electrical and Computer Engineering, Air Force Institute of Technology, Wright-Patterson AFB, OH. She received a Bachelor of Science in Geology, Magna Cum Laude from University of Miami, Coral Gables, FL in 1984 and a Bachelor of Science in Electrical Engineering, Summa Cum Laude from North Carolina State University, Raleigh, NC in 1986. Also from North Carolina State University, Raleigh, NC, she received a Master of Science in Electrical and Computer Engineering in 1987 and her Doctor of Philosophy in Computer Engineering (Artificial Intelligence) from Clemson University, Clemson, SC in 1995. Her research interests include artificial intelligence, intelligent computer generated forces, associate systems, distributed virtual environments, intelligent human computer interaction, and man-machine interfaces.

Eugene Santos Jr. is an assistant professor of computer science at the Air Force Institute of Technology. He received his B.S. in Mathematics and Computer Science (1985) and M.S. in Mathematics -- Numerical Analysis (1986) from Youngstown State University (1985) and subsequently completed a Sc.M. (1987) and Ph.D. in

Computer Science -- Artificial Intelligence (1992) from Brown University. His research interests include automated reasoning, neural networks, natural language understanding, expert systems, machine learning, operations research, probabilistic reasoning, robotic planning, temporal reasoning, combinatorial optimization, numerical analysis and parallel processing. Member, IEEE, ACM, Sigma Xi and AAAI.

Martin R. Stytz is an active duty Lieutenant Colonel in the U.S. Air Force serving as an Associate Professor of Computer Science and Engineering at the Air Force Institute of Technology. He received a Bachelor of Science degree from the U.S. Air Force Academy in 1975, a Master of Arts degree from Central Missouri State University in 1979, a Master of Science degree from the University of Michigan in 1983, and his PhD in Computer Science and Engineering from the University of Michigan in 1989. He is a member of the ACM, SIGGRAPH, SIGCHI, the IEEE, the IEEE Computer Society, the Engineering in Medicine and Biology Society, the IMAGE Society, the Virtual Reality Society, and the Society for Computer Simulation. His research interests include virtual environments, distributed interactive simulation, modeling and simulation, user-interface design, software architecture, and computer generated forces.



Moving Intelligent Automated Forces Into Theater-Level Scenarios

Randolph M. Jones, John E. Laird, and Paul E. Nielsen
Artificial Intelligence Laboratory
University of Michigan
1101 Beal Avenue
Ann Arbor, MI 48109-2110
{rjones,laird,nielsen}@eecs.umich.edu

1. Abstract

Intelligent synthetic agents have participated successfully in a series of military simulation exercises. However, their participation has been limited mostly to rather simple missions and situations, and they have been less successful in terms of participating in, and reasoning about, missions that require paying attention to an entire theater of operations and the concerns that come with such a global picture. From our experiences with theater-level exercises, we have identified a small number of categories for improvement of intelligent synthetic agents in this regard. In addition, we have devoted significant effort to implementing solutions to these concerns, and have successfully demonstrated synthetic agents that behave intelligently and flexibly in simulated theater-level operations.

2. Introduction

Intelligent automated forces have successfully implemented a subset of the behaviors required to model individual agents in combat engagements (Tambe et al., 1995). The goal of such forces is to increase the fidelity of training simulations by generating human-like behavior. Building such forces involves creating agent models that incorporate the knowledge and capabilities that humans use to achieve their missions.

The first implemented IFOR's successfully generated human-like performance for relatively simple missions and constrained situations. For example, Jones, Tambe, Laird, and Rosenbloom (1993) describe intelligent agents that participate in limited one-on-one air-to-air engagements. Rosenbloom et al. (1994) expanded the abilities of the agents to perform a small number of specific air missions, such as combat air patrols and sweeps. This work also implemented an initial capability for groups of agents to communicate and coordinate with each

other (e.g., flying in section).

We had the opportunity to evaluate these agents by participating in the STOW-E exercise. In general, the participation of IFOR's in their first operational military exercise met with success. However, participating in such an exercise made it obvious that there is really no such thing as an isolated mission that is independent of a larger theater of activity. Intelligent forces were able to fulfill their specific, limited roles in the theater-level exercise, but we were also able to identify a number of ways that we could increase the flexibility of the forces, so they would be more useful in theater-level simulation exercises.

The remainder of this paper first identifies three general categories of requirements for bringing intelligent forces into full-blown theater-level exercises. We then present a number of specific solutions we have developed to meet these requirements. These solutions enable the deployment of intelligent forces in theater-level exercises, and culminating with their participation in STOW-97.

3. Desired Capabilities

Our experiences in STOW-E highlighted a number of desired capabilities that would have made deployment of the intelligent forces much easier and more robust. These capabilities fall into three general categories, which we present here.

3.1 Agent Knowledge

The first category of capabilities involves enriching the knowledge base of individual intelligent agents, so they take theater-level goals and constraints into account while executing their particular missions. This helps the agents generate more appropriate and human-like behaviors when faced with a wide range of choices.

At the theater level, the agents must incorporate knowledge of the overall exercise into many of their local decisions. For example, for a BARCAP mission, it is not enough for an agent to know that it must fly a racetrack at a particular waypoint, oriented in a specific direction, and intercept any enemy aircraft that come close enough. In addition, the agent must know about the following types of things:

- How long to remain on station.
- Who the air traffic controller is, and how to interact with it.
- When and where to refuel.
- Specific, possibly complex commit criteria to begin an intercept.
- Keeping track of where friendly aircraft are, and who they may be intercepting.
- Which direction to force the intercept, based on what is being protected.
- Levels of acceptable risk, for deciding which tactics should be used during an intercept.

Another example of theater-level concerns for individual agent knowledge involves the ability to fly in packages for strike or close-air support missions. Strike packages consist of a number of individual missions, such as sweeps, suppression of enemy air defense, escorts, tankers, and the strike itself. However, all the agents executing these missions must have the capability to coordinate and communicate with each other. In addition, they must have knowledge of how to execute proper ingress and egress profiles, as well as the ability to interact with various command and control entities.

These capabilities represent a significant increase over those required to carry out specific missions (which are themselves non-trivial). In an intelligent synthetic agent, these capabilities translate directly into requirements for new knowledge. For the agents to operate successfully in a theater of war, they must incorporate a large amount of new knowledge that specifically addresses the theater-level concerns.

3.2 Intelligent Support Agents

The second category involves expanding the infrastructure for supporting intelligent agent roles in theater-level exercises. For our purposes, this includes building intelligent forces to model all the entities involved in the air combat portion of a theater of operations. Examples of agent behaviors that support theater-level activities are airborne

early warning, forward air control, refueling, and escorting strike packages. Each of these jobs can be performed by a human or an intelligent force, and they all contribute to the quality of decision-making by individuals in the theater.

As we hinted above, the core capabilities for intelligent agents enable agents to fly relatively autonomous missions, such as executing an intercept, or delivering ordnance to a ground target. In reality, however, these missions require many different types of support in order to be successful. At the very least, support can be provided by humans in simulators or real vehicles, but even in this case the intelligent synthetic agent must have knowledge of what these support roles are, and how to interact with them. At the other end of the spectrum, however, we would like all of the support missions to be implemented by synthetic agents as well. In order for these agents to be intelligent, they must incorporate complete knowledge of how to fulfill each support role.

3.3 Tools and Interfaces

The third category of capabilities involves developing tools and interfaces that support the use of intelligent forces in theater-level simulations. These tools should allow humans to specify scenarios and control agents as easily and flexibly as possible. Facilities exist for specifying the missions of semi-automated forces, and manipulating them during the course of an exercise in response to new demands for the training scenario. However, the existing interfaces do not apply well to groups of intelligent agents, which require interactions that are much more similar to those for human participants in the theater of operations.

The participation of our agents in the STOW-E exercise revealed two fundamental weaknesses in the tools we had to interface with intelligent synthetic agents. The first involved the interface for specifying mission and situation briefings for the agents. The basic interface allows us to specify scenario and mission parameters to each individual agent, but the process must be repeated for each agent. This is fine when agents are executing separate missions, and do not need to share much information. However, when theater-level concerns are introduced, there is much necessary information that should be shared among different groups of agents.

For example, waypoint names and locations generally hold across an entire exercise. A particular Airborne Early Warning aircraft will be assigned to interact with a number of flights of aircraft. All the members of a division of aircraft will share call signs and other information. Thus, when all the information for an exercise needs to be conveyed to all of the agents, there can be an enormous amount of redundancy. To address this problem requires a shift away from “agent-centered” mission interfaces, moving instead to an “exercise-centered” interface, which reflects a more flexible organization for mission information, and shares information among appropriate groups of agents.

The second interface weakness arose toward the end of the STOW-E exercise. Most of the exercise missions were scripted weeks in advance, and we were able to specify the appropriate missions for all of our agents so they could execute them on demand. However, toward the end of the exercise, it became clear that the administrators running the exercise wanted to change the script dynamically in response to changes in the training situation, or to explore variations of the initial exercise.

With standard synthetic forces, it is no problem to “pick up” the aircraft, move it to a different location, and assign it a new mission. However, with intelligent agents this is not so simple, largely because intelligent agents maintain a memory of events in the current mission, as well as knowledge structures representing their current awareness of the overall situation. If you try to “pick up” and move an intelligent agent, you will get a similar type of confusion that might arise if you suddenly teleported a human pilot to a different portion of the world. The agent will not know where it is or why. It will no longer have an accurate picture of its situation, because the situation will have changed. In addition, it will not necessarily be a simple matter for the agent simply to abandon its old mission and pick up a new one from some arbitrary starting point.

In order to address this situation, we require an interface that allows exercise administrators to interact with intelligent synthetic agents in a manner similar to how they interact with human participants in the exercise. When a human needs to be assigned a new mission—for example, to execute a BARCAP at a new waypoint—a controller must instruct the pilot by radio to knock off the current mission and fly to the new waypoint. Then

the pilot must fly there; the aircraft cannot magically transport. There are similar requirements for interacting with an intelligent synthetic agent. The agent must be instructed by a simulated radio with new orders, and then it must carry them out itself, in order to maintain a reasonable awareness of the world around it.

4. Progress and Results

We do not yet have fully autonomous, intelligent, synthetic agents participating in theater-level exercises. However, we have completed significant amounts of research and development in each of these three categories.

4.1 New Agents and Capabilities

The first two categories have not so much required the creation of new tools as they have a redoubling of our effort to acquire and engineer knowledge to make the intelligence of our agents as rich as possible. To begin with, we have built agents that fulfill a number of new roles to fill out the simulated theater of war. These include agents that perform the following functions:

- Tanking.
- Strike escort.
- Suppression of enemy air defense.
- Airborne Early Warning.
- Reconnaissance.

In addition, we have developed limited agent capabilities to provide a number of command and control functions:

- Land/Launch control.
- Strike control
- Tactical air control center.
- Direct air support center.
- Fire support coordination center.
- Forward Air Control (airborne and ground-based).

These command and control agents come nowhere near providing the full capabilities for each of their missions, but they provide enough support that they can fulfill simulated roles to enable our other air agents to execute their missions in a realistic manner.

Finally, we have enriched our basic agents with the knowledge that allows them to interact appropriately with each of these new agents. In addition, knowledge has been incorporated to allow

the agents to coordinate in packages for large-scale strike, interdiction, and close-air support missions. Finally, the knowledge for defensive missions has been updated so that agents can provide a more integrated defense, taking into account theater-level concerns, as well as coordinating intercepts and defensive assignments between different groups of aircraft.

4.2 New Mission and Command Interfaces

In order to address the need for improved interfaces to our agents, we have developed two new graphical interfaces, both of which greatly improve the flexibility and ease of interacting with the intelligent synthetic agents.

The first tool is an exercise editor for specifying all the information relevant to a theater-level exercise and translating it to parameters used by our intelligent agents. The editor organizes mission and scenario specifications into multiple levels: exercise, event, mission, and entity. Entity-specific information is tailored to the smallest groups of agents (divisions, sections, or individual vehicles). However, information common to a single mission or a single event (for example, an integrated strike) only needs to be entered once, even though it is then passed on as parameters to a number of individual agents. The highest level of the hierarchy contains information common to all agents over the entire course of the exercise. Coulter and Laird (1996) provide more complete details on this editor. Experience with the editor has shown that it is relatively easy to learn, and that it greatly improves the efficiency of specifying new exercises involving our synthetic agents.

The second tool we have developed is a graphical interface panel to allow simulated radio communication with the intelligent agents. The interface provides templates for semi-natural language phrases that the intelligent agents can understand and know how to obey. This allows an exercise administrator to give orders to the intelligent agents in a relatively natural style of discourse. Because the agents communicate with each other using the same language and the same simulated radio interface, they do not care whether they are receiving orders from a human or from another appropriate synthetic agent. When an intelligent synthetic agent receives such an order (for example, to assume a new CAP station or to vector to a particular bogey group), it can parse the message, obey

it, and begin executing the appropriate maneuvers (and changes in mission specification and situational awareness) to implement the order. This tool allows a measure of flexibility for administering synthetic agents in a simulated exercise, while maintaining their ability to fulfill the role of a human-like participant in the theater of war.

5. Conclusion

As discussed in this paper, we have approached the problem of bringing intelligent synthetic agents to theater-level exercises from a variety of directions. In October of 1995, we were able to participate in an ARPA-sponsored simulation exercise, called ED-1. This exercise provided an opportunity for us to evaluate our progress and compare it with our participation in STOW-E. Our experiences indicate that we have made strides in the right direction. It was much easier to develop the scenarios for the exercise, using the exercise editor. We were able to demonstrate a number of new capabilities in the intelligent synthetic agents. Finally, we have demonstrated the capability to change scenarios by giving agents new orders via the graphical communications panel. We plan to continue our efforts in these directions, eventually providing robust interfaces and intelligent synthetic agents for STOW-97.

6. Acknowledgements

This research was supported at the University of Michigan as part of contract N66001-95-C-6013 from the Advanced Systems Technology Office of the Advanced Research Projects Agency and the Naval Command and Ocean Surveillance Center, RDT&E division. The research presented here has benefited greatly from the efforts of Paul Rosenbloom, Milind Tambe, Karen Coulter, Frank Koss, BMH Associates, Inc., and the other members of the Soar/IFOR project.

7. References

- Coulter, K. J., & Laird, J. E. (1996). A briefing-based graphical interface for exercise specification. In *Proceedings of the Sixth Conference on Computer Generated Forces and Behavioral Representation*. Orlando, FL.
- Jones, R. M., Tambe, M., Laird, J. E., & Rosenbloom, P. S. (1993). Intelligent automated agents for flight training simulators. In *Proceed-*

ings of the Third Conference on Computer Generated Forces and Behavioral Representation, (pp. 33–42). Orlando, FL.

Rosenbloom, P. S., Johnson, W. L., Jones, R. M., Koss, F., Laird, J. E., Lehman, J. F., Rubinoff, R., Schwamb, K. B., & Tambe, M. (1994). Intelligent automated agents for tactical air simulation: A progress report. In *Proceedings of the Fourth Conference on Computer Generated Forces and Behavioral Representation*, (pp. 69–78). Orlando, FL.

Tambe, M., Johnson, W. L., Jones, R. M., Koss, F., Laird, J. E., Rosenbloom, P. S., & Schwamb, K. B. (1995). Intelligent agents for interactive simulation environments. *AI Magazine*, 16(1), 15–39.

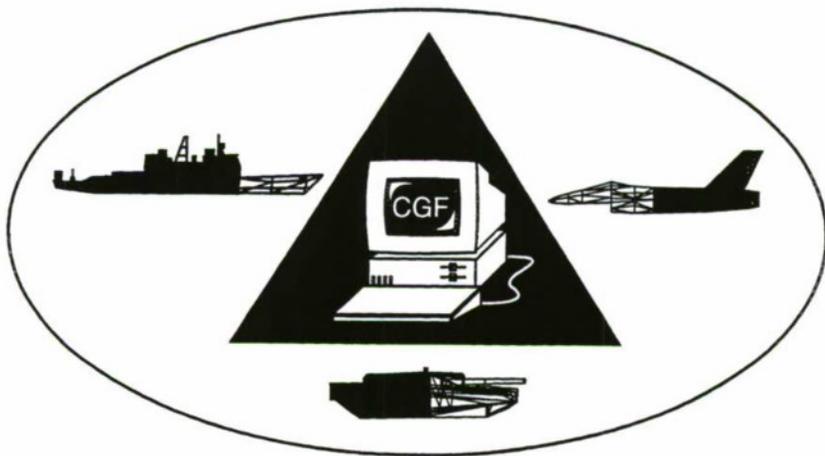
8. Biographies

Randolph M. Jones received his B.S. in Mathematics and Computer Science from the University of California, Los Angeles, in 1984. In 1987 and 1989, respectively, he received M.S. and Ph.D. degrees in Information and Computer Science from the University of California, Irvine. Subsequently, he served as a postdoctoral research associate in the Department of Psychology at Carnegie Mellon University and the Learning Research and Development Center at the University of Pittsburgh. He is currently an assistant research scientist in the Artificial Intelligence Laboratory at the University of Michigan. His research interests lie in the areas of intelligent agents, problem solving, machine learning, and psychological modeling.

John E. Laird is an associate professor of Electrical Engineering and Computer Science and the director of the Artificial Intelligence Laboratory at the University of Michigan. He received his B.S. degree in Computer and Communication Sciences from the University of Michigan in 1975 and his M.S. and Ph.D. degrees in Computer Science from Carnegie Mellon University in 1978 and 1983, respectively. His interests are centered on creating integrated intelligent agents (using the Soar architecture), leading to research in problem solving, complex behavior representation, machine learning, and cognitive modeling.

Paul E. Nielsen is an assistant research scientist at the Artificial Intelligence Laboratory of the University of Michigan. He received his Ph.D. from

the University of Illinois in 1988. Prior to joining the University of Michigan, he worked at the GE Corporate Research and Development Center. His research interests include intelligent agent modeling, qualitative physics, machine learning, and time-constrained reasoning.



Design of A DIS Agent, the AISim System: A progress report

Sakir Kocabas¹, Ercan Oztemel², Mahmut Uludag
and Nazim Koc

Marmara Research Center, Department of AI
PK 21, Gebze, 41470 Kocaeli, Turkey

email: (skoca,eomam1,mahmut,nazim)@yunus.mam.tubitak.gov.tr

1. Abstract

An intelligent system, AISim is being developed by the AI group at MRC, within the framework of multinational battlefield simulation project (EUCLID RTP 11.3). AISim is being developed to enable a simulated air target (an F16 plane) to behave intelligently in cooperation with other computer generated and man controlled air targets, in tasks and activities in CAP and Escort missions in defensive and offensive scenarios. The system's tasks include Navigation, Patrol, Escort, BVR and WVR Engagement, Air-to-Air Refueling, Disengage and Return-to-Base.

2. Introduction

The study of intelligent agents in real-time simulation systems has been one of the most challenging research topics in artificial intelligence (see, e.g., Jones, et. al. 1994). The primary purpose of such studies is to examine agent behavior in real-time environments and scenarios, and to prepare more realistic systems for training human operators for certain skills. Recently, extensive research is being carried out on intelligent agents operating in distributed interactive simulation (DIS) environments. The DIS environments enable to use a number of agents with different goals and behavior patterns in real-time scenarios (see, e.g., Oztemel & Kocabas, 1996; Laird, et al., 1995; Tambe, et al., 1995). DIS is mainly concerned with time and space-coherent, synthetic representation of real-world environments and interactions of operational entities in them.

The synthetic environment is created through real-time exchange of data units between distributed, and computationally autonomous simulation applications in the form of simulations, simulators and instrumented equipment interconnected through standard computer communicative services. The computational entities can be in one location or distributed geographically. A DIS system has the following characteristics:

- No central computer is used for event scheduling or conflict resolution.
- Autonomous simulation stations are responsible for maintaining the state of one or more simulation elements.
- There is a standard protocol for communicating ground-truth data.
- Receiving stations are responsible for determining what is to be perceived.
- Simulation stations communicate only changes in their state.
- "Dead-reckoning" algorithms are used to reduce overloads in processing communication data.

An intelligent agent consists mainly of three components: perception, cognition and action. Memory, reasoning, learning, understanding, planning, scheduling, and control are some of the basic characteristics of intelligent behavior. An agent equipped with these capabilities can receive information from its environment, organize its knowledge about the environment, evaluate situations, deduce conclusions, solve problems, and generate actions.

¹ Also affiliated with: Department of Space Sciences and Technology, ITU, Maslak, Istanbul, Turkey.

² Also affiliated with: Department of Industrial Engineering, SAU, Adapazari, Turkey.

The cooperation of DIS agents depends on the kind of tasks and activities they are expected to do, and the environment in which they operate. There may be three different types of tasks: 1) Agents may perform problem solving in a common domain, 2) agents may be working together to improve their individual performance, and 3) agents may be working together to improve the performance of the overall system they are designed for. In DIS systems the third type of cooperation is important as it concerns the question of dependency between agents. If an agent needs to communicate with other agents, it has to know the underlying model of these agents. Additionally, there has to be a standard data communication accessible by every entity within the overall system. Some data communication problems are solved by "dead reckoning" algorithms. Such algorithms estimate the future situations in the temporary absence of situational data, ensuring that the system is somewhat fault tolerant with respect to temporary communication failures. In a complex environment, knowledge used by an agent can be incomplete, and the goals of the agents might be conflicting (Jones, et. al., 1994). If an agent has conflicting goals, a set of heuristics or a classifier can be used to deal with the conflict. However, if different agents have conflicting goals, then there is a need for a negotiator to deal with this problem. The negotiator is an agent which defines the authority of information.

This paper describes the design of an intelligent agent, AISim, operating in a DIS environment. Our study focuses on the following problems in designing such agents:

- Rationality of agent behavior
- Agent cooperation and coordination
- Resolution of conflicts in agent goals and tasks
- Agent situation and behavior explanations
- Agent reusability.

The design history of AISim goes back to the design of its prototype RSIM (Kocabas, et. al., 1995). RSIM was a simple model operating in a 2-d space, with capabilities of learning its rules of behavior and explaining its behavior. AISim is a much more developed version with the capabilities of detailed situation assessment, action management and behavior explanation. In the following sections, first a summary of the design history of AISim is provided. Then the system is described in terms of its hardware and software structure. Next, AISim's methods and capabilities are discussed in comparison with other related systems. Finally, the paper concludes with a summary of the results.

3. System Development

The following procedure is employed in the development of AISim:

- Domain analysis to define the activities to be simulated in the application.
- Requirements analysis, to define the system's goals and functions.
- Global design analysis, to ensure that each specified goal is achieved by a set of functions.
- Detailed design, to guide the software engineers to code the system in accordance with the specified requirements.
- Software development which is the actual code generation process.
- Testing, verification and integration to DIS system.

Currently, this work has passed the prototype and design stages, and is now in the software development stage, in which AISim has been integrated with the underlying simulation system.

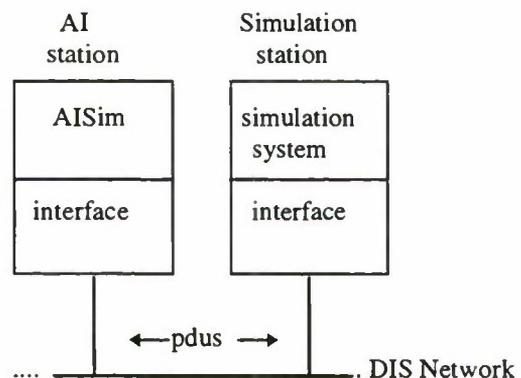


Figure 1. Hardware structure of the DIS system on which AISim runs.

4. System Description

The hardware structure of the DIS system on which AISim runs is shown in Figure 1. The system operates as networked to the simulation system in a DIS environment, where AISim runs on the AI station, and control its agent(s) on a simulation station connected to the same DIS system. The simulation station runs ITEMS¹ simulation system. The communication between the workstations is carried out by exchanging standard data units in the network, under InterSIM² a DIS network software.

¹ ITEMS is the product of CAE Electronics.

² InterSIM is the product of TTS.

As to the software architecture of the system, we have selected a hierarchical approach for the design of AISim, in which the system has four levels of goals:

- 1) Mission goals
- 2) Task goals
- 3) Subtask goals
- 4) Activity and action goals.

DIS scenarios require the definition of mission goals such as air interception and tactical air support. AISim has been designed for two different mission goals: Combat Air Patrol (CAP), and Escort to bombers. When the system's mission goal is defined as CAP, it is divided into a set of task goals such as navigation, patrol, and BVR combat. These task goals are further divided into a set of subtask goals such as trajectory guidance, weapons management, and evasion. The subtask goals in turn, are divided into activities such as firing and guiding a missile, performing an evasion maneuver, and turning towards a target. Activities are also divided into a set of simple actions such as changing heading, speed and altitude.

AISim's control structure supports the goal hierarchy described above. The system has two modules: Situation Assessment (SA) and Action Management (AM). The SA module monitors the situational parameters 10 times a second on average, by first selecting a set of situational parameters, calculates the situation, and sends a reduced set of situational indicators in the form of signals to the AM module.

The AM module itself consists of a set of operators in a hierarchy. On the top of this hierarchy is the Task Control Operator (TCO), which controls a set of task operators by deciding which task operator is to be activated under the current situation. Once a task operator is activated, this in turn, fires subtask and activity operators and rules. In this way, AISim directs its agent in the scenario in accordance with its assessment of the current situation.

AISim's TCO has the following operators which can become active in a CAP mission: Takeoff, Navigate, Patrol, BVR Engage, WVR Engage, Disengage, Air-to-Air Refueling (AAR), Return to Base (RTB), and Land. Each of these operators have a set of subtask operators which in turn have a set of activity operators, and finally each activity operator has a set of action rules. The task control operator of AISim currently has 23 rules for selecting task operators for

CAP missions. The total number of sub operators in these task operators is 52, which in turn have a small set of action rules and procedures. Figure 2 shows a section of AISim's mission, task, subtask and activity hierarchy. In this hierarchic control structure AISim supports the following intelligent agent characteristics: Situation assessment, action management and explanation.

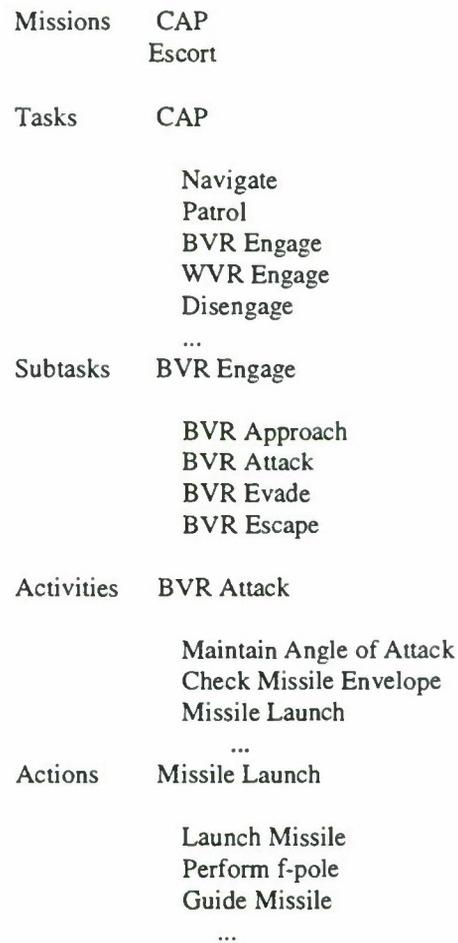


Figure 2. AISim's hierarchy of operators for mission tasks, subtasks, activities and actions.

5. Discussion

In this section AISim is discussed and compared with other related systems in terms of:

- Domain tasks
- System architecture (knowledge organization)
- Intelligent agent features
 - . Situation assessment (perception, cognition)
 - . Action management (cognition, action)

- . Robustness
 - . Timeliness
 - . Flexibility (e.g. reusability)
 - . Learning,
 - . Explanation.
- Performance in mission scenarios.

AISim has been tested in controlling an F 16 against ITEMS and man controlled Mig 29's and F 16s in various CAP scenarios. Tests on the system in escort scenarios are continuing. In CAP scenarios, the AISim agent (AIT) takes off, navigates to a patrol waypoint in a predefined desired engagement zone (DEZ), performs patrol in an elliptical orbit towards a given threat direction. When a threat approaches a certain distance, AISim's TCO passes control to BVR Engagement operator, and this in turn, to BVR Approach sub-operator, and so AIT leaves patrol and approaches its target in a certain angle. Within a certain range, BVR Attack sub-operator takes control of AIT, guiding it through to own missile envelope, while securing and maintaining radar lock until a certain range. This sub operator is also responsible for launching and guiding BVR missiles. Meanwhile, if a radar lock comes from the opponent in a certain range, control passes the BVR Evade sub-operator, which in turn, guides AIT into evasive maneuvers. Chaff throws and radar jams can automatically be taken care of by the simulation system ITEMS. During BVR Attack or BVR Evade, if AIT has entered WVR engagement range, then TCO passes control to WVR Engage operator which directs AIT in WVR attack, evade and escape maneuvers.

At all times, TCO checks the fuel and missile stocks of AIT. When AIT runs out of BVR and/or WVR missiles, control passes to Disengage and RTB operators depending on the tactical situation. When the fuel level of AIT is below a predefined level, and the mission is still on, TCO passes control to Escape and/or AAR operators, and AIT is directed towards an AAR point where it refuels.

The above is a brief description of AIT's behavior in which a good deal of details are omitted for reasons of the limitations of this paper.

The knowledge organization and control structure of AISim is based on the hierarchic homuncular control (HH) architecture (Kocabas, 1991). Unlike the sequences of operators of Soar-IFOR, in this architecture, AISim's operators are systematically divided into mission, task, subtask and activity operators as shown in Figure 2. This architecture provides effective search control in real-time

behavior. Accordingly, at any moment in its activity, the AISim agent can pass from one task (such as BVR Engage) to another task (e.g. Disengage).

The number of operators and rules of AISim are small, compared to the variety of tasks and activities performed by its agents in a scenario. There are two reasons for this:

- 1) AISim's HH control architecture has proved to be effective in partitioning the control of agent activities.
- 2) Many of the low level activities such as navigation to a waypoint and radar lock are carried out by the ITEMS simulation system.

Like Air-IFOR agents (Laird, et. al. 1995), AISim agents are isolated from the details of the underlying simulation environment, such as missile and plane dynamics, and sensor simulation. However, unlike Air-IFOR agents, AISim controls its agents created in a simulation station in the DIS environment, from a separate workstation connected to the same environment, using the data protocols of the DIS network software InterSIM. In other words, as opposed to Air-Soar systems which run in direct communication with its simulation system ModSAF on the same workstation, AISim runs independently on a separate workstation. Therefore its configuration is more general in terms of data communication and control than that of Air-Soar.

As to the intelligent agent features of the system, AISim's SA module reads the set of data on the dynamic and static simulation elements, and computes the parameters of the tactical situation from some of these data, and sends the relevant attribute-values to a message list to be read by the system's TCO operator. AISim reads about 60 different types of data (which are grouped in themselves), and sends about 15 types of data to the DIS network. The simulation system's clock cycle is 20 Hz. AISim's action management operators, as have been described above, are capable of guiding its agent in different tasks and activities. The current version performs well in 1-v-1 engagements, and has a simple set of prime opponent selection rules to deal with more than one opponent at a time. However, unlike Air-IFOR agents the system has not yet been developed for 1-v-2 and 2-v-2 air combat scenarios.

AISim tests shows that the system is robust in the sense that the system shows reasonable performance in different scenarios in 1-v-1 and 1-v-2 engagements. The system has also passed the timeliness criterion in its current form.

As to the flexibility criterion, AISim architecture has proved to be flexible enough in adapting to other missions (e.g., from CAP to Escort missions) simply by adding new task operators and a small set of task control rules in TCO. Unlike Air-Soar's procedure, in which this system uses a decision procedure to select operators according to the current situation by using a rule set for operator selection, in AISim task selection is done by its task control operator. One advantage of this architecture is that it enables to change the doctrines of the AIT more easily.

We had tested learning methods on our earlier model RSIM (Kocabas, et al, 1995) which learns action rules to perform meaningful maneuvers in 1-v-1 engagements. Learning methods have been applied in limited activities such as learning pure pursuit (Hommertzheim, et. al., 1991) and certain close combat maneuvers (Crowe, 1990). AISim's architecture allows it to learn task control and activity rules, but the system's search space is too large for effective control and action rules. For this reason, we have postponed the implementation of learning methods in AISim. On the other hand, many military missions and tasks are taught by instruction. Air combat maneuvers are also well defined both in tactics and geometrical paths and trajectories. However, this does not mean that learning is not feasible in such systems, particularly because of the use of new technologies in missiles and planes.

Behavior explanations is an important feature for computer generated agents, as it is useful to know both for development and training purposes, what the agent has been doing at a particular moment during its activities. Behavior explanations can be in the form of post-mission explanations (Johnson, 1994) or in real-time (Kocabas, et al.,1995). Like its predecessor RSIM, AISim explains its agent's behavior in real-time. The system's knowledge organization, particularly its task based hierarchy of operators into tasks, subtasks, activities and actions, facilitates the detailed explanation of its agent's behavior in real-time. Air-Soar agents also have explanation capability, but as post-flight explanations (Johnson, 1994).

The same knowledge organization also facilitates to include the description of agent goals and intentions beside simple behavior explanations. Goal directed explanations can be useful in monitoring the agent behavior more closely, particularly the agent's situation assessment capabilities. We intend to implement this feature in AISim. Under these

considerations, we believe that AISim has a more flexible knowledge organization scheme and control architecture than that of Soar which provides the basic knowledge organization scheme to Air-Soar systems.

As opposed to TacAir-Soar (Tambe, et al, 1995), AISim can in principle deal with multiple independent goals simultaneously. We are in the process of implementing this feature in the system. AISim can control more than one AI targets in a scenario from one station, although we have tried and tested only one so far.

Like Air-IFOR agents of Air-Soar, the AISim provides the following capabilities to AIT: situation assessment, following flight plans, performing patrol in reference to a certain waypoint and opponent direction, prime opponent selection, attack and missile management, evasion and escape, escort behavior and tactics, fuel management, disengagement, and coordinating with other agents in escort tasks. To these capabilities, own behavioral explanation and target behavior interpretation must be added.

On the other hand, compared with Air-IFOR agents, AISim agents have a limited range of mission simulations, as confined to CAP and Escort.

6. Summary

In this paper we have described the design of an intelligent system AISim, capable of performing tasks and activities in CAP and Escort missions. We have also discussed the system's knowledge organization and control architecture comparing with other related systems. AISim's architecture supports intelligent agent requirements such as situation assessment, action management, timeliness, flexibility and behavior explanation.

7. Acknowledgment

This work is supported by Marmara Research Center under the EUCLID RTP 11.3 complex air warfare simulation demonstration project.

8. References

Crowe, M.X. (1990). "The application of artificial neural systems to the training of air combat decision-making skills". *In Proceedings of the 12th ITSC.*, pp. 302-312.

- Hommertzheim, D., Huffman, J., and Sabuncuoglu, I. (1991). "Training and artificial neural network the pure pursuit maneuver", *Computer Ops Res.* 18 No.4, pp. 343-353.
- Kocabas, S. (1991). "Homuncular learning and rule parallelism: An application to BACON", *In proceedings of International Conference on Control - 91*, pp. 950-954.
- Kocabas, S., Oztemel, E., Uludag, M., and Koc, N. (1995). "Automated agents that learn and explain their own actions: A progress report", *In Proceedings of the 5th Conference on Computer Generated Forces and Behavioral Representation*. pp. 63-68.
- Laird, J.E., Johnson, W.L., Jones, R.M., Koss, F., Lehman, J.F., Nielsen, P.E., Rosenbloom, P.S., Rubinoff, R., Schwamb, K.B. Tambe, M., Van Dyke, J. van Lent, E., and Wray, R.E. (1995). "Simulated intelligent forces for air: The Soar/IFOR project 1995", *In Proceedings of the 5th Conference on Computer Generated Forces and Behavioral Representation*. pp. 27-36.
- Oztemel, E. and Kocabas, S. (1996). "Design principles for intelligent agents in distributed interactive simulation", *In Proceedings of SimTect-96*, 25-26 March 1996, pp. 103-106.
- Tambe, M., Johnson, W.L., Jones, R.M., Koss, F., Laird, J.E., Rosenbloom, P.S. and Schwamb, K.B. (1995). "Intelligent agents for interactive simulation environments", *AI Magazine*, Spring, 1995, pp. 15-39.
- Johnson, W.L. (1994). "Agents that explain their own actions", *In Proceedings of the 4th Conference on Computer Generated Forces. May 1994, Orlando, Florida*, pp 87-95
- Jones, R.M., Laird, J.E., Tambe, M. & Rosenbloom, P.S. (1994). "Generating behavior in response to interacting goals", *In Proceedings of the 4th Conference on Computer Generated Forces and Behavioral Representation*, pp 317-324

9. Authors' Biographies

Sakir Kocabas is the head of the AI Department at MRC and the project manager for EUCLID RTP 11.3 WP2. Dr. Kocabas has a PhD degree in Information Engineering. His research interests are in the areas of Real-Time Simulation, Machine Learning and Discovery.

Ercan Oztemel is a researcher at the AI Department of MRC. Dr. Oztemel has a PhD degree in Artificial Intelligence. His research interests are Simulation, Real-Time Knowledge Based Systems, Inductive Learning and Neural Networks.

Mahmut Uludag is a researcher at the AI Department of MRC. Mr. Uludag has a Masters of Science degree in Mechanical Engineering, and is a PhD student at ITU. His research interests are AI Applications in Real-Time Simulation.

Nazim Koc is a researcher at the AI Department of MRC. He has a Masters of Science degree in Symbolic Computation, and is a PhD student at ITU. His research interests are Symbolic Computation, Parallel Logic Programming and Machine Learning.

Session 2b: Uses of CGF

**Brooks, U. S. Army, AMSAA
Berkowitz, The Mitre Corporation
Craft, UCF/IST
Metzler, LB&M**



Computer Generated Forces (CGF) Assessment

Wilbert J. Brooks and Marguerite M. Dymond
U.S. Army Materiel Systems Analysis Activity
Aberdeen Proving Ground, Maryland 21005-5071
e-mail: wbrooks@arl.mil

1. Abstract

Computer Generated Forces (CGFs) are software driven forces whose tactical behaviors/decisions are made by human commanders (semi automated forces) or automated algorithms (automated forces). CGFs were developed to support Army applications in three modeling and simulation domains. Distributed Interactive Simulation (DIS) applications have been demonstrated in all three domains. CGF developments to support DIS fall into two categories: new developments and modifications to existing CGFs.

The CGF Assessment evaluated seven CGFs to provide a basis for an Army investment strategy. The CGF Assessment addressed four questions:

1. What are the current and planned CGF capabilities?
2. What are the CGF characteristics?
3. Is the CGF credible?
4. Which DIS domain applications can be satisfied?

This paper presents the CGF Assessment approach and summary of results and the Army's CGF investment strategy.

2. Introduction

Computer Generated Forces are software driven forces whose tactical behaviors/decisions are made by human commanders (semi-automated forces) or automated algorithms (automated forces). CGFs were developed to support Army applications in the three modeling and simulation domains: Advanced Concepts and Requirements (ACR); Research, Development, and Acquisition (RDA); and Training, Exercises, and Military Operations (TEMO).

3. DIS CGF Development History

Distributed Interactive Simulation (DIS) proof of concept was demonstrated for unit training by the Simulation Network (SIMNET). SIMNET linked simulators together in a virtual environment with a semi-automated force (SAF) developed to represent both the threat and friendly units. Since SIMNET

proof of concept for unit training, DIS applications in all three modeling and simulation domains were demonstrated using variants of the SIMNET SAF. As a result of the SIMNET proof of concept for training the Army initiated the Combined Arms Tactical Trainer (CATT) program to develop the next generation of Army training systems to support combined arms training for units at the team through battalion task force level. The first phase of the CATT program is the Close Combat Tactical Trainer (CCTT) development. CCTT supports training for armor and mechanized infantry units. Manned simulators support the primary training audience. CCTT SAF fills out the battlefield with Opposing Forces (OPFOR) units and adjacent, supporting, or tethered Blue Forces (BLUFOR) units. CCTT is scheduled for fielding in 1997.

Modular SAF (ModSAF) development was initiated by the Advanced Research Project Agency (ARPA) in 1992 as a replacement for SIMNET SAF for DIS RDA and ACR applications and to support SAF advanced distributed simulation research. ARPA and the Simulation, Training, and Instrumentation Command (STRICOM) continue to sponsor its development and application in ACR, RDA, and TEMO domains. STRICOM's Battlefield Distributed Simulation-Developmental (BDS-D) Advanced Technology Demonstration (ATD) is enhancing entry-level representation for Army systems and verification and validation through the Anti Armor (A2) ATD. ARPA is sponsoring the development of an Air Force, Navy, and Marine Corps ModSAF as part of the Synthetic Theater of War (STOW) program. Due to the accessibility of the source code, ModSAF provides the foundation for the advanced distributed simulation research in the following areas: Intelligent Forces (IFOR), Command Forces (CFOR) SAF, synthetic environment phenomenology (e.g., clouds, smoke), real-time information transfer, networks, development of a High-Level Architecture (HLA), Synthetic Environment Data Requirements Information System (SEDRIS), and the next generation of DIS protocols.

CCTT SAF and ModSAF were designed to run in real-time, be DIS compatible (DIS compliant (send and receive DIS protocol data units) and coherent in time and space with respect to other simulations). STRICOM, PM CATT, and ARPA initiated planning to develop an interoperable (DIS compatible with consistent physical and behavior models that ensure a fair fight) CCTT and ModSAF capability.

Several existing CGFs are being modified to run in real-time and be DIS compliant and compatible:

a. Interactive Distributed Early Entry Analysis Simulation (IDEEAS) variant of Battlefield Environment Weapon System Simulation (BEWSS). BEWSS is a high fidelity engineering simulation that models critical performance characteristics for precision guided weapons in a realistic battlefield environment. It was developed by the Missile Command (MICOM) for trade-off analyses comparing different sensors, guidance, and missile designs in a degraded environment; performance comparisons of different terminally guided and smart weapons in a degraded environment; small unit smart weapon force mix analyses; and optimum employment studies for smart weapons. IDEEAS has two main objectives: 1) to create a DIS compatible version of BEWSS, and 2) to make BEWSS an interactive tool.

b. Interactive Tactical Environment Management System (ITEMS). ITEMS was developed by CAE Electronics, Montreal, Canada, for the Army Research Institute as an outgrowth of CAE's in-house capability as a simulator developer. Since ITEMS was designed to run in real-time to provide a virtual environment for simulators, its architecture supported making it DIS compliant and compatible. ITEMS is designed to support engineering analyses of rotor and fixed wing aircraft, including mission equipment and sensor systems in a realistic environment (including countermeasures). Today, ITEMS is also used by the Research, Development, and Engineering Centers (RDEC) for aviation, armaments, and tanks; industry; and foreign military facilities around the world.

c. Janus linked to DIS (JLINK). JLINK is a recently completed research project that developed a DIS compatible version of Janus (i.e., linked through an interface called World Modeler (WM)). It has been demonstrated with manned simulators and ModSAF. Janus was developed by Lawrence Livermore National Laboratory (LLNL) in the early 1970s. In 1983, LLNL transferred the source code and system design of Janus 1.0 to the Training and Doctrine Command (TRADOC) Analysis Center (TRAC).

TRAC has improved Janus through enhanced functional representations and a more robust operating systems (UNIX). Janus 5.1 is installed at over 50 sites worldwide where it is primarily used for training, combat development analyses, and research and development. TRAC is planning to continue research with JLINK with formal configuration management planned in FY98.

d. Joint Conflict Model (JCM). JCM was developed by LLNL as a derivative of the Janus simulation to support joint staff training exercises. Janus was expanded to represent a brigade task force supported by air and naval operations. Subsequently, JCM was re-designed using an object-oriented approach for its data structures and added a DIS protocol interface.

e. Joint Tactical Simulation (JTS). JTS is also a Janus derivative developed by LLNL. The current JTS is a re-engineered and significantly enhanced merger of Urban Combat Computer Assisted Training System (UCCATS) and the Security Exercise Evaluation System (SEES). JTS is a unique high fidelity simulation of dismounted combat in an urban environment. JTS is currently used for officer training and operational planning by U.S. Army Europe (USAREUR) and U.S. Army Special Operations Command (SOCOM), site security training and analysis by U.S. Army Southern Command (SOUTHCOM), and security training, analysis, and evaluation by Department of Energy (DOE) National Laboratories. Joint Warfighting Center (JWC) plans to merge JTS and JCM into a single simulation.

At a briefing for the Army senior leadership (charged with oversight responsibility for modeling and simulation to support acquisition) on ModSAF development status in October 1994, the need to assess these seven CGFs to provide a basis for developing an optimum Army CGF investment strategy for DIS was identified. The Army Material Systems Analysis Activity (AMSAA) was tasked to develop a plan to evaluate alternative CGFs because MSAA was managing the A2ATD and participating in the development and verification and validation of all CGFs except JCM and JTS. AMSAA formed a CGF Assessment Working Group composed of knowledge people within the Army (Army Materiel Command, TRAC, and National Simulation Center), Institute for Defense Analysis, the Mitre Corporation, Carmel Applied Technologies, and Illgen Simulation Technologies, Inc.

4. Purpose and Approach

The purpose of the CGF assessment was to evaluate the alternative CGFs for all Army DIS domains: ACR, RDA, and TEMO.

The CGF Assessment addressed four questions:

1. What are the current and planned CGF capabilities?
2. What are the CGF characteristics?
3. Is the CGF credible?
4. Which DIS domain applications can be satisfied?

Current and planned CGF capabilities were evaluated on the basis of two evaluation criteria: battle force representation and simulation modeling features. Battle force representation assesses the ability of the CGF to represent different types of weapon systems/military equipment for all services (Air Force, Army, Navy, Marine Corps); levels of military organizations (platoon through Division); Command, Control, Communications, and Intelligence (C3I); behaviors of entities and units; tactics and doctrine. Command entities and their ability to communicate with crews through the Command and Control Simulation Interface Language (CCSIL) were also assessed. Simulation modeling features characterize simulation models and data bases used to represent system performance and the environment.

CGF characteristics were evaluated on the basis of system architecture; simulation execution; system ergonomics; operation, maintenance, and expandability; and scenario development. System architecture addresses both hardware and software architecture. Hardware requirements were assessed for several scenarios. Software architecture addresses run-time architecture, software development environment and standards, system update rate and load balancing, hardware platforms that run the software, and DIS network interface.

CGF credibility was evaluated based upon verification, validation, and accreditation completed and planned. Long term configuration management was also considered.

5. CGF Requirements

While the CGF Assessment developed comprehensive evaluation criteria to assess capabilities, characteristics, and credibility, there are key criteria in each area that can be considered as discriminators among CGF alternatives to meet future CGF requirements. Required capabilities include flexibility in modeling unit behavior (i.e., semi

automated (player control) or validated automated), physical algorithm fidelity (using standard performance models for most applications and engineering models for unique applications requiring this level of detail), ability to represent the joint battlefield, and a high resolution environment (time of day, atmosphere, countermeasure for the entire electromagnetic spectrum).

Required characteristics include the ability to run in real-time, DIS compatibility, compliance with the High Level Architecture (HLA)-interfaces and object model templates for linking of simulation environments, and using an object management approach. Credibility requires verification, validation, and accreditation of the CGF for applications and a government controlled configuration management process and access to the source code.

6. Objective CGF Architecture

Before presenting summary comparisons of the seven CGFs capabilities, characteristics, and credibility, an objective CGF architecture that uses an object management approach (OMA) is described. OMA describes a software architecture for composing CGFs to meet applications from interworking component objects. Objects are defined in terms of attribute data and operations they make public. All objects require some services. Two or more (but not all) objects share facilities. Some objects have unique attributes and operations they make public (not common with any other object). Examples of services required by all objects are time and exercise management. Examples of facilities shared by two or more objects are physical models. An example of a unique object may be a weapon on one system (e.g. laser). Objects in a CGF (unique, shared, and required) are connected by an object request broker using an interface definition language.

7. Comparison of CGF Capabilities, Characteristics, Credibility

Comparison of required capabilities indicates that only three CGFs offer flexibility in modeling unit behaviors (semi automated or automated): CCTT SAF, ITEMS, and ModSAF. A higher level of unit behavior automation (CFOR) is also being developed in ModSAF as part of the STOW program. BEWSS uses scripted battles and is developing a semi-automated capability. All the Janus model derivatives (JCM, JLINK, and JTS) are only capable of modeling unit behavior through human

intervention (semi automated) or replay of scripts from previous Janus battles. All CGFs use standard performance models (or derivatives). BEWSS has implemented engineering models for missiles and smart artillery, and ITEMS has implemented engineering models for aviation systems (including weapons). JCM is the only CGF that has a joint battlefield representation. ModSAF is developing a joint battlefield representation as part of the STOW program. BEWSS and ITEMS have implemented a higher resolution environment representation for the systems engineering models. CCTT SAF plays terrain interactions and their impact on mobility at the highest level of resolution. JTS has a unique representation of urban terrain. Finally, a dynamic environment is being developed in ModSAF as part of the STOW program.

Comparisons of CGF characteristics indicate that only CCTT SAF, ITEMS, and ModSAF were designed to run in real-time and be DIS compatible. Only CCTT SAF and ModSAF are being developed and will demonstrate HLA compliance. In addition, ModSAF will be modified as part of the STOW program to demonstrate an initial object management approach.

CGF credibility depends upon verification, validation, and accreditation. CCTT SAF has the most comprehensive VV&A plans and configuration management process. The government owns or currently has unlimited rights to the source code for all CGFs excepts ITEMS. Although ITEMS is a proprietary code CAE is willing to negotiate sale of the code to the U.S. government.

8. Conclusions

The CGF Assessment concluded that:

1. BEWSS, JCM, JLINK, and JTS are not robust CGF solutions for DIS because they do not have automated behaviors, and they were not designed to run in real-time or be DIS compatible.
2. CCTT SAF, ITEMS, and ModSAF have the potential to meet all domain applications with additional investments. Integration of interoperable ModSAF and CCTT SAF functionality into a flexible, extensible SAF architecture that exploits distributed object management approaches and is HLA compliant should meet all future Army CGF requirements with the least additional investments.

3. BEWSS, ITEMS, JCM, and JTS will continue to meet unique CGF requirements (engineering trade-off analyses, joint operations staff training, and dismounted infantry in an urban environment) until integrated ModSAF and CCTT SAF are available with these capabilities. Unique functional capabilities in these CGFs should be identified and incorporated into the ModSAF and CCTT SAF architecture (if necessary).

4. The Army needs a ModSAF configuration management process that institutionalizes ModSAF V&V implemented by BDS-D/A2ATD. STRICOM (DIS Program Manager) has the lead for developing this process.

9. Army CGF Investment Strategy

The Army investment strategy is to integrate interoperable ModSAF and CCTT SAF functionality into a flexible, extensible SAF architecture that exploits distributed object management approaches and is compliant with the High Level Architecture being developed by DMSO. The short term ModSAF and CCTT SAF integration goal is to develop and demonstrate an interoperable ModSAF and CCTT SAF capability in STOW 97. This strategy leverages Army, ARPA, and DMSO investments in ModSAF, CCTT SAF, STOW, HLA, and Advanced Distributed Simulation research. The ModSAF and CCTT SAF integration will be managed by the CATT Program Manager.

The CCTT SAF and ModSAF integration program is built around a series of experiments and technical assessments, which progressively assess key issues in interoperability and integration of capability. The major issues affecting SAF integration are terrain interoperability, command and control, network communication, and architectural design.

10. Biography

Mr. Brooks is Chief of the Simulation Branch at the Army Materiel Systems Analysis Activity and the Army's A2ATD Program Manager.

Ms. Dymond is an Operations Research Analyst at the Army Materiel Systems Analysis Activity.

Considerations for the Use of Entity-based Simulations for Tactical Decision Making Training

Jack P. Berkowitz
The MITRE Corporation
49185 Transmitter Road, Bldg. 626, San Diego, CA 92152
jack@mitre.org

1. Abstract

The LeatherNet system provides a unique opportunity for the use of CGF by operational commanders. In this article, the development of the LeatherNet training system is reviewed, highlighting the training and human computer interaction principles which have guided development. The system incorporates modules that perform synthetic force simulation, speech recognition, visualization, and decision aiding. Next, the beginning efforts at performing structured training effectiveness evaluations are reviewed, along with the shortcomings of this evaluation.

Two critical elements are highlighted in connection with using Computer Generated Forces (CGF) for decision making training. First, the fidelity of the CGF behaviors must be relatively high, despite the relative abstractness of some of the trained concepts. Second, although often tacked on last, the user interface to the CGF must be considered up front when designing CGF applications, including the design of the actual simulated forces.

2. Introduction

As part of the agreements reached between the Defense Advanced Research Projects Agency (DARPA) and the United States Marine Corps, a development was initiated that would benefit both DARPA's ongoing Synthetic Theater of War (STOW) program and the Marine Corps Air Ground Combat Center (MCAGCC) Twentynine Palms' training mission. This cooperative development would result in a capability that could be used to actively train Marine Corps users for a specific series of exercises at MCAGCC.

The development is being conducted by the Naval Command Control Ocean Surveillance Center, Research Development Test and Evaluation Division (NRaD) on behalf of the DARPA Synthetic Forces Program Manager. NRaD is assisted in the development by a series of contractors located throughout the country. MITRE supports the development by running the integration and training site at MCAGCC Twentynine Palms and through other engineering in San Diego and other sites.

The LeatherNet system was devised for two purposes: 1) to provide a mechanism for subject matter expert feedback regarding synthetic force behavior for the Marine Corps component of STOW; and 2) to provide a real-use exploratory system for the Marines cycling through MCAGCC for Combined Arms Exercise (CAX) training. Included in this development was the implementation of a research laboratory and test bed at MCAGCC for the development of the training system.

This paper is organized in three sections. First, a recap of the system requirements and design is presented. Next, the preliminary efforts at evaluation of training system effectiveness, and efforts to improve these results, are presented. Last, a discussion of needed training concepts and components is provided.

Note that the specifics of the LeatherNet implementation are available from other sources (Osga and Murray, 1994; Berkowitz, 1995a; Berkowitz, 1995b). The focus on this paper will be on the rationale for developments and not specifically on implementation considerations; however, some references to the prototype implementation are provided.

3. Development of the Training Concept

The LeatherNet system is a group of computerized tools for the primary purpose of performing command decision training and tactical mission pre- and post briefs. Central to the LeatherNet system is the concept of *wargaming*, in which users can establish scenarios and conduct battles using friendly forces against unfriendly forces. Essentially, LeatherNet provides tools to allow a Marine Corps user to perform wargaming using simulated individual, mechanized, and airborne forces, and to manipulate the ongoing simulated battle through a series of visualization tools.

3.1 User

The intended user of LeatherNet is a Marine commander at the company commander or battalion level of command (0-3+). The Marine can perform multiple tasks using the system, including plan

assault and defense missions on a simulated battlefield, run these planned missions using simulated troop behaviors, and review/critique previous simulated and real battles using the assembled computer tools.

3.2 System Requirements

The system is designed around a series of basic training and human computer interaction requirements, all focused on the central development concept presented above. In the next few paragraphs, the essential system requirements are presented. The first few requirements are major driving functions for the system design and are expanded here, with the additional requirements presented in a reduced format for brevity.

3.2.1 Knowledge of Results.

Essential to the positive transfer of training is the effective use of Knowledge of Results within the system (Jacobs, *et al.*; 1993). Including adequate methodologies for experimentation of tactics (so-called “bands and sequels”) and arranging the system to rapidly track and analyze results can aid in the trainee understanding what went wrong with his plan as well as what was correct. The timely presentation of results can be beneficial both for recalling information not readily evident, as well as indicating changes that might be easily implemented. If presented within the same session, the trainee might be able to continue to work using his adapted strategy, and thus successfully complete the simulated mission.

Included in this concept is the proper application of multimedia tools to provide adequate and appropriate knowledge of results to the Marine Corps user. This is an important concept to ensure positive training transfer as opposed to null or negative training. For example, if the concept being trained involves the three-dimensional relationship between a commander’s ground units and supporting indirect fire, then it might be best to present this information in a three-dimensional format, as opposed to text tables or scores.

3.2.2 Training Needs

A primary goal of this guideline is to select and prioritize information and design techniques towards providing meaningful training (Goldstein, 1986; Hays and Singer, 1989). For example, a range of informational items may be displayed describing the status of an individual SAF entity. The selection of which information should be selected is determined by what is more meaningful to the Marine commander being trained. Given that systems have limited display space and techniques, an appropriate

trade-off would be to display the current and intended orders for the entity, as opposed to the DIS protocol entity number. In addition, information can be prioritized according to the importance at the specific time in the simulation, as this may also prove useful for training.

3.2.3 Unified User Interface

Despite the array of computer tools and hardware systems employed, the user must see a unified view of the LeatherNet system. The central component of this view is provided by the human computer interface (HCI) of the CommandVu system. Essentially, CommandVu provides a three dimensional representation of the simulated environment, and provides method for the user to interact with the CGF entities. This view is augmented by two dimensional map and information displays to provide additional tactical and planning information. The interaction of the user will be restricted to the CommandVu system, with necessary interaction to other LeatherNet components, including Marine Corps Synthetic Forces (MCSF), performed using the CommandVu Input Manager.

The reason for this restriction is two-fold. First, the users of this system are Marine Corps commanders who, although undoubtedly proficient in several computer systems, are using LeatherNet for command training. This training is provided once or twice a year at the Marine Corps Air Ground Combat Center (MCAGCC), 29 Palms. The various tools for the command training, including MCSF, each have unique user interfaces, and are undoubtedly different than the common user interfaces the Marines will have on their personal computers. Therefore, using a common user interface for all LeatherNet tools will simplify the process of learning a computer system for the Marines — the purpose of the system is to produce better trained military commanders, not better trained computer users.

Second, conventional command training takes place in the field; at 29 Palms, two sample maneuver ranges are Range 400 for troops and Delta Corridor for mechanized units. In the field, the commander can visually acquire both friendly and enemy targets, as well as use other sensors for detection. Commands are largely verbal, with responses from units provided back via radio. The LeatherNet system mimics this relationship through three dimensional representation and voice input and feedback. There are a series of complex cognitive skills combined for command decision making, and by maintaining some of the environmental constants for the commander, hopefully negative training transfer effects due to using an “artificial” system can be minimized.

3.2.4 Synthetic versus Real Environment

The environment created for LeatherNet is a synthetic environment which provides representations of the real-world using computer generated imagery and sound. However, the intent of the system is not to create an absolute replica of the real world. The level of environmental fidelity reproduced by the system should be determined in reference to the identified training needs of the Marine Corps user community.

The distinction allows for a range of possibilities in terms of interface design. A primary implication is the ability to add decision aiding overlays in the graphical environment. For example, an graphical enveloped may be added to the display which represents the fly-out patterns for weapons. This representation is not available on a real training range; however, the knowledge of the weapon ranges may assist in overall decision training, and therefore can contribute valuable information. Similarly, the visual detail provided on entities appearance, the environment, and other information can be manipulated to reach an optimal level between the absolute reality and the needs of the trainee.

3.2.5 Natural Input Methods and Language

Driven from the training needs requirement is the selection of input and output methods which will not interfere with the intent of the training mission. The techniques used to control entities, and for that matter the training environment, should match those that are used in the real world to the degree possible. If not, it is possible that negative training effects could be encountered, with commanders getting very good at manipulating a computer, but actually learning bad or incorrect troop movements due to using the computer.

3.2.6 Timing of Events

Command decision making, particularly in combat, is an extremely time critical process. The information received from the field must be relayed quickly and clearly, and subsequent orders must be transmitted and received efficiently. The LeatherNet system must similarly provide a realistic response relationship in terms of processing orders and providing feedback to the Marine using the system.

3.2.7 Dynamics of Combat Environment

The battlefield is a very dynamic environment. Despite the best planning and strategy, the commander must constantly change and alter orders in response to developments, both positive and negative, in the battle. One key to executing the strategies is knowing that the changed orders have been received and are being followed by the subordinate troops. Similarly, the LeatherNet user

interface must incorporate adequate flexibility and feedback cues so as to enable the commander to direct the battle as he/she would in the real world.

3.3 System Description

Given the adoption of the requirements above, a series of design activities were initiated, including limited training needs assessments, projective task analyses, story boarding, and paper user interface prototypes. Based on the preliminary design work, a system was designed and is currently being implemented geared towards the MCAGCC CAX user. The major components of the LeatherNet system are described briefly in the next sections.

3.3.1 Synthetic Forces

For each mode of operation, the heart of the LeatherNet system is the Marine Corps Synthetic Forces (MCSF) that are depicted on various displays. These forces are computer simulations of actual troop and vehicle behavior, termed *entities* in the computer modeling field. The behavior of entities is based on extensive knowledge acquisition on actual troop and vehicle movements, and the translation of this knowledge into computer models of the specific component. The focus of the MCSF project is the development of infantry or individual combatants for STOW, although additional vehicles and weapons platforms which are Marine Corps specific are also being developed (for example, the Amphibious Assault Vehicle).

MCSF is a computer program which can be used to manipulate and control the behavior of entities on a simulated battlefield. In this respect, MCSF is the heart of the LeatherNet system, as it is the tool that drives the balance of the simulation system. MCSF is built on top of an existing military simulation program, ModSAF, which allows for the control of tactics, troop response, environment, enemy position, and many other aspects of the simulated battlefield. Again, the additions for MCSF include enriching the capabilities for individual combatant entities to fight, receive commands, and react to various stimuli, including MCSF is capable of being run in a stand-alone configuration; however, LeatherNet provides a series of visualization tools and decision aids which can greatly enhance the training and simulation environment for the user.

3.3.2 Visualization

The primary tool provided is the CommandVu visualization environment. CommandVu provides a three-dimensional representation of the entities on the simulated battlefield, and allows the user to move throughout the battlefield to view and control movements from different vantage points. In

addition, CommandVu provides a series of graphical, auditory, and intelligent decision aids to assist the user in planning troop movements, in learning and predicting troop response, and in following maneuvers and decisions *post hoc*. Most importantly, all user interaction with the LeatherNet system, including the ModSAF simulation engine, is provided through CommandVu.

3.3.3 Terrain Analysis

What this feature allows for is the combining of additional tools with ModSAF in order to strengthen the training potential of the system. The Terrain Evaluation Model (TEM) can provide quick analyses of terrain data for use in the system. Operating off of military terrain data bases, TEM provides line-of-sight and weapons coverage analysis capabilities. The eventual goal is to have the outputs of the terrain analysis displayed in both two and three-dimensions.

3.3.4 Natural Input Methods

Additionally, DARPA sponsored Voice Recognition tools can be used to pass messages to ModSAF to allow the Marine user a more natural method of interaction with the simulated entities. These tools allow for speech recognition and natural language understanding of verbal commands issued from commanders and commands for controlling the simulated environment. Essentially, the speech recognition system allows the user to use "radio-speak" for command and control of the simulated forces.

In addition, methods for combining pen-based gesture and voice recognition for transmitting graphical plans to the simulated forces are being developed. The method of interaction for these tools include the drawing of attack positions and avenues of approach onto a tactical map display, and then transmitting these "pictures" to the forces. This method attempts to replicate the use of paper maps and acetate overlays commonly used for small unit planning.

To effect these inputs, several techniques are employed. Display devices include:

- Helmet mounted displays
- Large projection displays arranged as a walk through environment.
- Three dimensional sound
- Speech generation.

Input devices include:

- Speech recognition
- Keyboards and virtual on-screen buttons.
- On screen controls (tape displays)
- Joysticks and other tracking devices.

4. Evaluation of the Training Concept

The initial implementation of the LeatherNet system proceed based on several iterations of design analyses and reviews. However, throughout the entire development process, essential questions regarding the effectiveness of the system to prepare company commanders for CAX exercises continued to surface.

These questions were complicated due the relatively long development cycle required of the system. Although portions of the system had been in existence separately in various forms, the combination of the systems for training had not been tried previously. In addition, the complexity of the software system required long lead times for translating requirements into stable and somewhat functional platforms.

In the fall of 1995, it was decided to perform a semi-controlled experiment using actual CAX participants as subjects. This test would occur approximately 14 months after the initiation of the development effort, and a full two years before the scheduled completion date of the system.

The focus of the testing for the development team was to identify if the training concept was feasible given the constraints of the CAX schedule and requirements that was administered at MCAGCC.

The emphasis from the Marine Corps was to identify those parts of the system that were useable to the CAX process immediately, and begin to phase these developments into the on-base training cycle as they made sense. It should be noted that at no time was the initial testing construed to be an evaluation of the work being produces, i.e., this was not an acceptance test for modeling and simulation by the Marine Corps.

4.1 Test Conduct

The initial focus of the use of LeatherNet was for the mission analysis and briefing of the Range 400 exercise which occurs during the CAX. The R400 exercise is an infantry company dismounted attack on a series of lightly fortified positions "manned" by fictional enemy within a box canyon. It is a live-fire exercise for the Marine Corps, and includes the use of indirect fire weapons (mortars), heavy machine guns, and shoulder launched missiles (DRAGON).

For a Marine company commander, this exercise represents several of the critical decision making tasks which he must employ in combat. Included in these decisions are the use of suppressive fire to support movement, the employment of supporting forces such as engineers, to augment regular assets,

and the management of resources such as ammunition and personnel. Successful execution of the mission resembles a well executed football play, with a continuous flow of troops up the canyon. Unsuccessful execution is marked by a lack of movement, the running out of ammunition, and generally disrupted performance. It should be noted that the development team members, many of whom had no experience with Marine Corps land maneuvers, quickly learned to spot successful and poor R400 performances.

The exercise is administered by the Tactical Exercise Evaluation and Control Group (TEECG), a permanent resource at MCAGCC who operate the CAX process. This group had been functioning as the subject matter experts for the MCSF developments, and would function as the eventual administrators of the LeatherNet system. The choice of the R400 scenario was also beneficial as it was one of the major development scenarios for the MCSF individual combatant development to date.

Breaking down the R400 exercise further, the execution consists of six distinct elements, including two obstacle breaches, the movement of indirect and supporting fires, and the clearing of three different trench systems. The execution of each of these elements in sequence provided a framework for the employment of the LeatherNet system.

4.2 Test Implementation

Soon after initiating the test planning, it became obvious to both the Marine Corps and development team representatives that a complete use of the LeatherNet system would not be possible, nor would complete use of the synthetic forces be achievable. In terms of CIF, several of the behaviors needed to complete an entire company-level mission sequence were judged to be relatively immature. Furthermore, the visualizations to support some aspects of planning, including the linkage to terrain analysis, were not complete. This situation was not surprising considering the development schedule and the thrust for early evaluation, but it did impact the use of different tools in the evaluation.

It was decided to approach the use of the system in a lecture format course with a TEECG representative acting as an instructor for a course in basic infantry operations. The course would focus on several elements from the METT-TS-L concept (Mission, Enemy, Terrain, Tactics, Time, Space, Logistics), and particularly the Intelligence Preparation of the Battlefield (IPB) (U.S. Army, 1994). Due to the maturity of the different tool sets, the initial course would emphasize the use of the TEM and

CommandVu systems for terrain analysis, with the use of CGF limited at first.

The Marine Corps desired to employ some method of using a control and experimental group for the test, sending half of company commanders into the LeatherNet system for a lecture course, with the control group receiving a similar briefing, but only using the 1:50K resolution maps that are standard issue. For the two CAX rotations used for the evaluation, this represented three company commanders in each group. The exposure for both groups would be two hours, with similarly topics covered.

It should be noted that this was far from a structured training effectiveness experiment. Due to the logistics of units participating in the CAX process, it is difficult if not impossible to get equivalently balanced performers assigned to different study groups. Similarly, as part of existing range orientation practices at MCAGCC, all commanders are allowed to walk the actual terrain with safety officers, and the Marine Corps was understandably hesitant to suspend this practice for an initial system test.

The other major difficulty was in the quantification of results from the training. In a classical training effectiveness study, a series of transfer measures are collected from both objective and subjective data to derive some indication of transfer and corresponding transfer effectiveness (i.e., dollar spent for amount of incremental improvement). Collect measures from both the control and experimental group and essentially a rough measure of transfer can be deduced.

Although seemingly straightforward for the R400 problem, the difficulty lay in the fact that company commanders do not actually "fail" on R400. This situation does not constitute a grade inflation for the commanders, it is just a more subjective assessment between the TEECG, the superior officers in the unit, and the commander himself. There are a multitude of factors that are uncontrollable in the exercise, including weather, the state of readiness of the unit, basic experience level, and so on which complicate any evaluation. However, attempts were made to formulate some qualitative measures centered around the six R400 phases, and used in the subsequent analysis.

4.3 Test Results

The test results break into two elements, those concerning LeatherNet system use, and those concerning specific CGF use and interaction. Note that the collection of data did not follow a strict

evaluation protocol. Although a more rigorous evaluation procedure was proposed, the Marine Corps chose to use a questionnaire approach, again in reference to the relative early stage of system development (Berkowitz, 1996). Further transfer studies are planned as development continues.

4.3.1 System Use

The results of the initial testing indicated that the system, although limited in its application, did have some benefits to the user population. Subjective comments included the improved selection of approach and target points, the use of intelligent terrain analysis tools, and the improved use of the lecture/classroom tools by the instructor (an unintentional improvement was the perceived improvement in the curriculum that the entire CAX process underwent). TEECG subjective assessments of the student's performance did not differ between the control and experimental group, however, given the nature of the software status and the constraints on the evaluation, this result was not unexpected.

Most important for the LeatherNet development was the feeling within the TEECG that the system did have some usable components even in the current state, and that the TEECG subsequently decided to phase in all aspects of the system into regular CAX training rotations. This resulted in the development of a revised R400 curriculum featuring the LeatherNet system's use during the preparation for mission stages. LeatherNet use continued into the 1996 CAX schedule, with additional features introduced into the system as of the April, 1996 CAX rotations.

4.3.2 Use of CGF

This latest development is most critical to the concept of entity based CGF used for training. As noted previously, the use of CGF in the Fall 1995 evaluations was limited. This was primarily due to the limited maturity of the CGF behaviors. Although these behaviors were considered sufficiently developed for successful demonstration in the STOW Engineering Demonstration I activities conducted in the Fall of 1995, the TEECG felt that they were not to the level required for active student use.

Specifically, the TEECG felt that the CGF should be able to conduct elements of all six required phases (breaching, machine gun set-up, etc.) for the use by *students* for their active training. Anything short of complete and reasonably accurate behaviors could prove disrupting to the student's ability to grasp concepts that needed to be expressed. For example, a critical concept in the R400 training is the use and control of machine guns as a base of fire for maneuvering troops. It would be necessary for the student to be able to accurately control the machine

guns produced by CGF, and to have the squads behave according to standard operating procedures for the exercise, so as not to distract the student from learning to control the fires as part of an overall movement. **The concern was that if a student had to focus in on a specific piece of CGF that was not acceptably functioning, then he would be distracted from the overall situation of the exercise.** Interacting with the behavior of the CGF was the method of controlling the CGF --- if the interaction was not a natural method, **then the trainee would similarly be drawn into an artificial mode and away from the actual intended training situation.** These two concerns form the basis of the training concepts and issues to be presented in section 5.

It should be noted that as more mature CGF behaviors are coming on line within MCSF, they are being phased into the training program at MCAGCC. For example, as of the April CAX, the use of cover and concealment by CGF infantry squads is of sufficient "fidelity" that the TEECG is allowing some limited use of their behaviors for students to time movements

5. Entity-Based Training Concepts and Issues

Based on the experiences from LeatherNet developments to date, and specifically the use of LeatherNet tools for company commander training at MCAGCC, several issues can be put forth concerning the use of entity-based simulated forces for training. These issues are somewhat complex and have several layers of variables associated with them, however, an attempt is made to constrain them to the low level commander use being experienced at Twentynine Palms. Several additional concerns may be more or less important with the increasing echelon levels of the training audience.

For the LeatherNet system, two areas of concern arise: 1) the functionality of the CGF behaviors, and 2) the methods for human interaction with the CGF. Although discussed separately below, both factors are inextricably linked when considering system implementation. Also discussed will be a more structured method of training evaluation and assessment that would prove useful in improving system design.

5.1 Functionality of CGF behaviors

MCSF behaviors are being developed to support the missions of the Marine Corps component for the STOW exercise. The R400 exercise is almost a perfect match to these requirements because the six mission phases executed are also present in the

preliminary mission assignments for infantry in the STOW scenarios.

However, at the levels of the STOW training audience (JTF and Component commanders), small problems with CGF performance will not significantly affect commander decision making. The same is not true at a lower echelon level, and in fact, can effectively render the training useless. **The critical point is that at lower echelon levels, CGF behaviors to support decision making must be exact and precise, even if the decision making concepts are relatively abstract such as resource management.**

For example, the situation where the CGF do not adequately employ machine guns means that the balance of the mission can not be executed. For illustration, suppose that the CGF expend ammunition at a rate 10% faster than the company commander desires, simply due to a slightly differing standard operating procedure. To most CGF development teams, this rate of fire problem would not seem extreme, and although it could be fixed in time, would probably not prevent the system from falling outside of some acceptable bounds. However, in response, the company commander must make alterations to his plan during the scenario to account for these events. This presents the situation where the simulation no longer adequately represents the real world to the trainee, thereby preventing positive transfer of training. An argument can be made that these situations happen in the real world and that the commander should be able to adjust to these constant management demands, but this fails to be sustainable across all possible situations.

Similarly, CGF behaviors must be inclusive of all missions that might be encountered during a specific exercise. The key to decision making training in the R400 scenario is the successful stringing together of actions or phases that are usually practiced in solitary. One Marine Corps advisor likens these CAX exercises to the "big game" as opposed to "blocking and tackling drills". Thus, successful completion of the complete exercise requires all aspects of the complete system to be functioning to some degree of adequacy. The execution of a mission without the capability to perform an obstacle breach would be similar to attempting to forward pass without blocking in the backfield.

5.2 User Interface to Support CGF Interaction

Whereas the first observation on CGF functionality is probably not novel to experienced developers, it is probable that the second observation has not received enough attention by developers. **Specifically, no matter how sophisticated the CGF behaviors, the**

system is limited by the relative efficiency of the user interface to the CGFs.

A shortcoming of the system implementation as of the date of testing was the degree of CGF control allowed via natural user interaction modes. Despite the fact that the CommandVu system was designed around the Marine Corps user, and has been constructed on the base with constant input, the jump between normal operation methods and the level of interface to CGF entities is large. The most significant change is the degree to which Standard Operating Procedures must be specified directly to CGFs, where as operational forces would have pre-knowledge or comfort with these concepts. Users easily became involved in the intricacies in specifying engineer level parameters and behavior sequences. Therefore, subsequent development efforts in the user interface are featuring the use of standard operating procedures, natural language processing, and other "regular" interaction methods.

This concept is also extendible to higher echelon training exercises. Complex behavior assignments are sometimes necessary for the completion of taskings that can be expected from higher echelon commanders. For example, in MCSF, a commonly expected order that should occur in STOW is to have infantry embarked on helicopters that would in turn take off from an amphibious carrier. Once disembarked from the aircraft, that infantry would then be tasked with a series of missions appropriate to their echelon. In terms of simulation, this sequence represents a series of complex mission assignments and procedures. Failures in the user interaction with this system would affect the eventual training audience either through incomplete execution of simulated entity behaviors, incorrect execution, or poor timing.

The LeatherNet system attempts to counter these problems through the use of natural interaction methods for CGF, including visualization, speech recognition, natural language understanding, and gesture technologies. Together, these methods seems to be effective in minimizing some learning of the systems, but more importantly, help to lower the apprehension of users to CGF. This is an important point as CGF continues to mature and become accessible to different types of users.

However, despite adoption of novel interfaces and methodologies by the LeatherNet project onto an existing CGF architecture, the system will probably continue to be limited and require expert "assistance" from LeatherNet workers. The root of the solution to these problems lies in the development of the CGF entities themselves. The CGF entities have been

designed for performance to exacting performance specifications, but not to specifications for their control by human operators. Just as C4I systems are now being designed with the user interaction as a central focus (for example, dynamic automation assignments based on user mode), the simulations which stimulate these C4I systems must consider the human user before coding can begin (for example, should there be a SOP file such that the user can rely on it?)

This methodology is starting to be adopted within the MCSF development strategy. As new behavioral developments are initiated, so are the language and gesture developments to support these behaviors. As MCSF incorporates speech interaction, efforts are made to keep speech capabilities in pace with ongoing behavioral developments. Although there is admittedly some lag in such developments, this methodology has had some success, such that baseline versions of MCSF for distribution are speech, and in the future gesture, enabled.

The beginnings of such user-centered design is also evident in the Command Forces (CFOR) project for STOW, through which the infantry platoon and company commander entities are being constructed. In the design and analysis stage, the actual communication processes between echelons are being documented and implemented in the form of Command and Control Simulation Interface Language (CCSIL) messages.

A suggested improvement to CGF development would be the completion of a user system interface section during the Knowledge Acquisition and Engineering phases of the development process. Similar methods have been used successfully in the development of complex decision aiding systems (such as flight management systems in aircraft) and in other complex system developments. Such a section would not only contain the specific messaging intended for CGF commands, but also a notion of user specification of behaviors and parameters.

5.3 Methods to Assess Training Effectiveness

A major difficulty in the work to date has been the adoption of a standard methodology for the evaluation of training effectiveness for LeatherNet. Although standards of commander performance do exist, these are only loosely applied within the R400 evaluation, as the collection of objective parameters during the live fire exercises is difficult, and as noted previously, the evaluation of performance is highly individualized for each commander.

Two developments should improve this situation. First, under separate programs, the Marine Corps is beginning the development of a range instrumentation system at R400. This instrumentation system could allow for the collection of objective data including direct measures of performance such as ammunition expenditure, and more indirect measures, such as time to complete certain mission phases.

A second effort is the development of rating scales for use in evaluating company performance on R400. These scales have been developed in reference to the specific battle drills and phases executed in the R400 mission, and are targeted in answering whether positive transfer of training has occurred through LeatherNet system use. However, as these are not current operational evaluation matrices used by the TEECG in the field, their active use for CAX participants remains questionable. The hope is that other units stationed at MCAGCC or visiting to execute R400 outside of the normal CAX schedule could be used as participants in such evaluations.

6. Acknowledgment

This work was performed under contract DAAB07-96-C-E601. The work presented here represents the efforts of many government and contractor personnel working on the STOW program and LeatherNet. In particular, the author would like to thank Mr. Frank Carr of MITRE for his critical review.

7. References

- Berkowitz, J. (1995a) *Technical and Scientific Report: Human Computer Interaction (HCI) Design Guidelines and Concepts for LeatherNet*. San Diego, CA: Galaxy Scientific Corporation for the Naval Command Control and Ocean Surveillance Center.
- Berkowitz, J. (1995b) *Technical and Scientific Report: LeatherNet Human Computer Interaction (HCI) Functional Design*. San Diego, CA: Galaxy Scientific Corporation for the Naval Command Control and Ocean Surveillance Center.
- Berkowitz, J. (1996) *Test Report: LeatherNet Human Systems Interface Design*. San Diego, CA: Galaxy Scientific Corporation for the Naval Command Control and Ocean Surveillance Center.
- Goldstein, I. (1986) *Training in Organizations: Needs Assessment, Development, and Evaluation*. Pacific Grove, CA: Brooks/Cole Publishing Co..
- Hays, R. and Singer, M. (1989) *Simulator Fidelity in Training System Design*. Springer-Verlag.

- Jacobs, R., Crooks, W., Crooks, J., Colburn, E., Fraser, R., Gorman, F., Madden, J., Furness, T., and Tice, S. (1993) *Behavioral Requirements for Training and Rehearsal in Virtual Environments (ARI-TR-9130-000-03-93)*. Orlando, FL: Army Research Institute.
- Osga, G. and Murray, S. (1994). *Preliminary Design Document. Concept of Operations: CyberView Human Computer-Interface*. San Diego, CA: Naval Command, Control and Ocean Surveillance Center, RDT&E Division.
- United States Army. (1994) *Intelligence Preparation of the Battlefield (FM 34-130)*.

8. Author's Biography

Jack P. Berkowitz is a Senior Human Systems Interface Engineer with the MITRE Corporation. His current responsibilities include the design of the user interface and training protocols for the LeatherNet training systems, as well as issues related to the usability of systems in the STOW exercise. Previously, he was involved in the design and evaluation of aviation security systems, flight deck communications, and aviation fire and rescue platforms for both the FAA and airline industry. Mr. Berkowitz has a M.S. degree in Industrial Engineering and Operations Research (Human Factors) from Virginia Tech. His research interests are in the areas of Complex Automation Systems and Virtual Environments.



Testing Future Weapons Systems Using CGF Systems

Michael A. Craft and Clark R. Karr
Institute for Simulation and Training
3280 Progress Dr., Orlando FL 32826
mcraft@ist.ucf.edu, ckarr@ist.ucf.edu

1. Abstract

Techniques developed to use virtual simulations to evaluate new or proposed systems (vehicles or weapons) are discussed. An experimental application of the techniques is described. Some of the practical problems encountered are discussed.

The techniques described are suitable for evaluating proposed systems, system modifications, and, possibly, tactics.

Many important decisions can be made by using virtual simulation based evaluations early in the acquisition process without endangering people, disturbing the environment, or huge expenditures.

2. Evaluation Techniques

The evaluation of a weapons system (or a simple vehicle or a tactic) using the techniques described requires a suitable CGF system along with people who know how to use it and software engineers capable of enhancing it. Experts are required to develop test scenarios and evaluation metrics for the experiments. Project members are needed to examine the statistical significance of the experiments.

The Vehicle Under Test (VUT)¹ must be compared with some baseline, either an existing vehicle or a competing variant. For the purposes of discussion, this paper is couched in terms of two competing variants.

2.1 Personnel Requirements

The work described is more than just the development of software. Scenarios used to test vehicles should be designed by people who understand military scenarios.

¹ For simplicity, the remaining discussion refers to vehicle tests and the Vehicle Under Test (VUT) although the technique has much wider application.

Similarly, it is inappropriate to have software engineers "invent" metrics for determining the outcome of the experimental scenarios. It may be that "off-the-shelf" scenarios or metrics are available and can be employed, but such plans should be reviewed by Subject Matter Experts (SMEs).

2.2 CGF System Requirements

Many, perhaps most, virtual simulation packages that comply with the Distributed Interactive Simulation (DIS) protocol are suitable for this technique. Non-DIS systems can also be used, providing they are capable of generating the needed (externally visible) information. The discussion here is in terms of the analysis of DIS traffic.

The CGF system will almost certainly require modifications to support the variants to be evaluated. Whether this is done by adding a new vehicle type for each variant or by building a distinct system for each variant is irrelevant. Other modifications may be required to support test scenarios. These matters (and the difficulty of implementing them) are in part a function of the quality and completeness of the CGF system selected. Modification may also be necessary to support the evaluation scenarios.

A primitive CGF system or a vehicle implementation without accurate vehicle characteristics (speed, correct weapons modeling, valid damage information and modeling, etc.) will yield untrustworthy results.

2.3 Scenario Development

A series of scenarios must be developed to exercise the VUT in a manner consistent with its expected use. Many scenarios may be required to represent a sufficiently rich test environment. Scenario selection is a substantial determinant of the quality of the outcome.

Each scenario should be customized for each VUT since vehicles with different capabilities will almost certainly use different tactics. It is unfair to use "generic" tactics if the VUTs should use different tactics.

2.4 Evaluation Metrics

A set of Measures Of Effectiveness (MOEs) must be established, preferably before any scenarios are run. It is probably necessary for the MOEs to be scenario dependent, although this makes the evaluation phase somewhat more complex.

In order to automate the evaluation process the MOEs need to be in terms of quantities that are visible through the (DIS) protocol. Many entity properties can be seen or deduced from network traffic, including the entity's health, location, velocity, and heading.

2.5 Experimental Variation

Although it would seem that all that is left is to run each variant through (its version of) each scenario and to compare the results, the situation is not that simple. For one thing, a scenario may not (and should not for these purposes) play-out the same from run to run: any single test run is just one battle in a family of possible battles for a fixed scenario.

Unless the scenarios are very simple, there is no way to be sure that a given run is representative of the family of possible outcomes. Many runs must be done and the MOEs applied to each in the hope of finding the mean MOE for a scenario-variant pair. It may be necessary to have some automated perturbation of the scenarios or CGF system².

Perturbation can be accomplished by re-seeding the random number generator (if one is used), moving vehicle's start points (slightly) from run to run, or changing other factors that do not distort the scenario design, but which might affect its modeling.

2.5.1 Sample Size

Statistical techniques (analysis of variance to compare two means of independent samples) can be applied to find the "superior" vehicle. However, the experimenter must be aware of, or control, the precision of the result.

The first set of runs should be used to estimate the mean and standard deviation for the MOEs (or the overall metric). Using these results the number of runs

² The process is analogous to selecting one element of a family of solutions to a differential equation by selecting an initial condition and then finding neighboring solutions by perturbing the initial condition.

required to distinguish the versions (for a given confidence) can be estimated.

2.6 Data Collection and Analysis

For each scenario, a number of trials are carried out for each variant. Statistical techniques are applied to the resulting MOEs to determine which variant is superior for the given scenario. This paper discusses an application of the techniques required.

Because results are likely to be mixed (some MOEs in some scenarios indicate variant-1 is superior but other combinations indicate variant-2 is superior) decisions will have to be made as to which scenarios and which MOEs are more important. Weights for each scenario and each MOE should be determined before experimentation is begun, and then a "decision" will be made by building a weighted average based on all the MOE results.

Whether such a mechanistic determination is acceptable is up to the experimenters. If many MOEs are used with many scenarios and a mixed result appears, endless debates are possible as to which VUT is superior.

In any case, these techniques are suitable to generate one or a raft of MOE results. If experiments are to give "an answer" scenarios and individual MOEs need to be combined.

3. Experiment Overview

The test case described here compared two variants of a proposed Advanced Amphibious Assault Vehicle (the Marine's AAV) running under ModSAF, a computer generated forces product developed by Loral³. A complete experiment design was done, and an analysis of the comparative value of the variants completed.

It is impractical and unnecessary to present the full experimental details here. Rather, examples in key areas are outlined. For a full description, refer to [Craft, 1995].

³ ModSAF is used for a variety of applications. It is used for experiments by BDS-D, A2ATD, and LOSAT. The National Guard uses ModSAF to support training exercises with manned simulators. It is used as an architectural prototype for the design for the Close Combat Tactical Trainer (CCTT) SAF system [Vrablik, R. and Richardson, W. 1994].

3.1 AAV Variants

The only *modeled* difference is that the model of Variant AAV-X has no Javelin mounted while the second variant, AAV-J, has a Javelin mounted on 1/3 of the vehicles. Other differences were not modeled, but this variation is by far the most important distinguishing characteristic between the two AAVs.

3.2 Accommodating the CGF

A daylight setting was specified for each mission with no clock time specified. This allowed the use of existing ModSAF behaviors. No weather or environmental conditions were created or supported. The scenarios were designed to avoid firing weapons over the surf zone since the Surf Zone is not a recognized ModSAF terrain type.

ModSAF uses Line-of-Sight (LOS) triggering as a fundamental mechanism; the system constantly checks LOS to determine if it has the ability to engage, report, etc. The scenarios were laid out to avoid immediate LOS between opposing forces. Scenario play allowed forward movement (Movement to Contact) and LOS acquisition (Enemy Contact); after that the ModSAF modeled behavior determined the sequence of interaction.

3.3 Terrain Selection

The terrain selected must be rich enough to support meaningful scenarios, it must be available in a supported (computer) format, and other conditions may come into play (such as the availability of military standard maps). The details of selection will vary from experiment to experiment. As a matter of information, the AAV experiment was carried out using two databases; one for Hunter-Liggett and the other a Korean TDB.

3.4 Scenarios

Two scenarios were developed by an SME to provide realistic and doctrinally correct interaction between the modeled players.

For the sake of illustration, one part of one scenario is briefly outlined here (each scenario contains various "sub-scenarios").

3.4.1 Sample Scenario Design

The "raid mission," in brief:

AAVs are to take out a SCUD. Reconnaissance indicates an OPFOR force to the North and South of the expected SCUD location. The AAVs leave a Mortar team on the beach while the remaining forces split North, East, and South (the N and S forces intending to set up blocks to protect the Eastern forces). OPFOR forces are encountered in the North and South, and battles ensue. The Eastern force climbs through switch-backs, encounters OPFOR forces, fights, and destroys the SCUD.

Complete scenario design rationales were developed, and a simplified Operations Order was written. A notional Marine Expeditionary Unit (MEU) was created along with an Amphibious Readiness Group (ARG) to support the operations order. AAV DRPM SMEs assisted with the scenario details.

In the Raid scenario the transition from the plan to the simulation in ModSAF required additional manipulation to replicate the human control measures found in a "live" exercise.

Many compromises were made to allow for the limitations in the CGF system and, in some cases, the CGF system was modified to accommodate the scenarios.

For example, the AAVs, once ashore, followed routes laid out to reach target positions, or move to contact. Initially movements were controlled based on time. This proved troublesome as changes in route caused changes in timing (leading to frequent scenario re-design and re-entry). A scenario modification was made to allow the use of military "Tactical Control Measures" similar to those used on operations overlays. This caused problems because the control measures had to be in LOS of the units for them to effect their control. A ModSAF modification was made to allow the control measures to be non-LOS based.

Cooperative behaviors embedded in ModSAF sometimes performed erratically. Vehicles would apply a higher priority to maintaining vehicle intervals (as they traveled) rather than returning fire. Modifications were made to the scenario, the ModSAF priorities, or the control measures to assure that the planned behavior was exhibited.

This table shows the timing, vehicles involved, and events modeled in the AAV Raid mission's east battle. The scenario recorded for analysis omits the ocean run-in and initial maneuvers. Adding these elements would offset times but has no significant impact on the

experiment proper (the run in was accounted for by the analysis tool).

East Battle at the SCUD Position

Time	Vehicle	Event
-2:15	All	Resume ModSAF
0:00	East AAV Platoon	Start moving in column formation along the East road at a maximum speed of 40 kph
0:07	East AAV Platoon SCUD BMP2 Platoon	See OPFOR vehicles at the SCUD site and begin Hasty Occupy Position See East AAVs and begin to move to line formation for Attack by Fire
0:20	SCUD BMP2 Platoon	In line formation along their battle line
0:26	AAAV's or BMP2s	First shot of this battle fired (exact time and vehicle varies)
1:05	East AAV Platoon	In line formation along their battle line
1:35	East AAV Platoon	Destroy the SCUD (exact time varies)
6:45	All	End

3.5 Measures of Effectiveness

The measures of effectiveness (MOEs) used are based on information gleaned from DIS traffic. Work done in this area is independent of ModSAF code and implementation. Methods developed in this experiment can be used with any DIS compliant CGF system.

Further, the software for gathering information and analyzing results was built in parallel with other work (serial development dependencies were greatly delayed).

3.5.1 Marine MOEs

The MOEs were built using section 3 (Measures and Methodology) of Advanced Amphibious Assault Vehicle (AAAV) Supplemental Analysis, Volume 1, Final Report (dated May 2, 1995). While the measures outlined there are not suitable as written, they yielded insight into what the Marine's consider when defining MOEs. The paper identifies 4 general, high level, MOEs:

1. Win Quickly
2. Win Decisively
3. Dominate the battlespace
4. Minimize casualties

To illustrate the complexities involved in automating the MOEs the rational used for "win quickly" and "dominate the battlespace" are outlined here.

3.5.1.1 Win Quickly

Win quickly is defined in terms of defeating orange breakout (time at which the last Orange battalion escaped) and the task force arriving at its objective (establishing blocking positions).

These are not suitable for the experimental scenarios, so "victory" is approximated. In one case, the destruction of a key entity is used as an indication of victory (denoted as "entity based victory" or EBV). In the other scenario victory is deemed to have been achieved when the last entity damage takes place (deemed "damage based victory" or DBV). DBV would be difficult to recognize in a live scenario, but it is not a problem for a logged scenario.

EBV is well suited to scenario-1 (the objective is the destruction of the SCUD) and this MOE is given considerable weight in scenario-1. The situation is not as clear cut in scenario-2 and so this MOE was not given a large weight for scenario-2.

3.5.1.2 Dominate the Battlespace

1. Seven quantities are listed for this MOE in the reference. The measurements are specified to be taken when combat intensity is "at a reasonably static state." This is far too vague for a computer analysis, but the scenarios used are simply allowed to run to completion and the measures are applied then. The MOEs include such things as Loss-exchange ratio (battalions), Orange/Blue and Orange battalions lost.

3.5.2 Experimental MOEs

An examination of the Marine MOEs shows only a few key quantities need to be tracked to allow computation of the experiment's MOEs. These building blocks are used to build key MOEs which are then normalized (re-mapped to a range of 0 to 1) and combined as a weighted average to produce a single MOE.

3.5.2.1 MOE Building Blocks

This table shows key quantities gleaned from the data logs by the analysis tool. These are the basis of building the experiment's MOEs. These quantities are given mnemonics to simplify their use in equations.

Description	Blue	Orange
Time until the key entity is destroyed	EBV	N/A
Time to last damage	DBV	N/A
Initial personnel count	B_IPC	O_IPC
Initial vehicle count	B_IVC	O_IVC
Initial force value	B_IFV	O_IFV
Final personnel count	B_FPC	O_FPC
Final vehicle count (Neutralized vehicles are not counted.)	B_FVC	O_FVC
Final force value	B_FFV	O_FFV

Based on the MOE building blocks key ratios are computed which tie directly to the Marine's MOEs. Here are the ratios used:

Description	Name	Computation	MOE Reference
O neutralized	OVN	$(O_IVC - O_FVC) / O_IVC$	Win decisively
Initial force ratio	IFR	B_IFV / O_IFV	Win decisively
Residual force ratio	RFR	B_FFV / O_FFV	Win decisively
Final Force Ratio	FFR	RFR / IFR	Win decisively
Loss-exchange ratio (vehicles)	LXV	$(O_IVC - O_FVC) / (B_IVC - B_FVC)$	Dominate Battle
Loss-exchange ratio (force)	LXF	$(O_IFV - O_FFV) / (B_IFV - B_FFV)$	Dominate Battle
Loss-exchange ratio (personnel)	LXP	$(O_IPC - O_FPC) / (B_IPC - B_FPC)$	Dominate Battle
Blue casualties	CAS	$(B_IPC - B_FPC) / B_IPC$	Minimize Casualties
Surviving Blue strength	SSTR	B_FFV / B_IFV	Minimize Casualties

Only four of the denominators can be zero under the assumption that all scenarios begin with at least 1 vehicle, which carries at least 1 person. Zero denominators may cause RFR, or any of the loss-exchange ratios, to be undefined. These special cases are handled as part of normalization.

Not shown is time to victory (EBV or DBV depending on the scenario) as that is not a ratio. The name "VIC" is used for the time to victory. In scenario-1 VIC represents EBV, whereas in scenario-2 VIC represents DBV.

3.5.2.2 Normalized MOE Ratios

It is problematic to build an MOE average using quantities with different ranges and so the values should be normalized. A typical technique is to map

the worst value observed (for a scenario using both variants) to 0 and the best value to 1.

It is possible that the time to victory between the variants will prove to be only a few seconds apart in the worst case and yet, with normalization, this could be artificially magnified. In some sense the normalization should map "poor" values to 0 and "good" values to 1.

The normalized form of X is spelled "N_X."

The eight individual MOEs are combined using a scenario dependent weighted average.

4. Experiment Implementation

The two scenarios designed for this experiment are intended to present the VUT in roles consistent with its expected use. They were created with the benefits and within the limitations of the CGF testing environment, ModSAF. Two variants of a new vehicle were added to the testing environment to study the feasibility of using virtual simulation for test, evaluation, and comparison of new or proposed vehicles.

4.1 ModSAF Modifications

Significant modifications to ModSAF were needed to support the vehicles under test and the test scenarios.

No amphibious vehicles exist in ModSAF 1.5.1 or DIS 2.0.3. The addition of this type required a ModSAF vehicle hull type capable of moving on both land and water and a definition for the amphibious domain (plus specific vehicle instances) in DIS protocol terms.

Along with a standard set of vehicle capabilities (e.g., routing, sighting and enemy vehicle engagement) the vehicle models need other distinguishing functionality such as the ability to rise in the water during acceleration and plane.

Some more details of problems encountered with the CGF system are covered in section 6.1, but for illustration, consider submergence.

ModSAF models the water surface but not the land on which the water sits. This is common to many, and probably most, CGF systems and is a function of the terrain representations in general use. The popular terrain representations are polygon based, and the terrain polygons have no thickness and consequently water has no depth.

To overcome this, a configurable beach inclination angle was added to the amphibious hull definition. When an amphibious vehicle crosses from a land polygon to a water polygon, the vehicle descends into the water polygon at this angle, until a maximum depth is reached. Similarly, when an amphibious vehicle is in water and approaches land, it climbs back onto land using the same slope.

4.2 Data Analysis

The scenario analysis is based on DIS traffic. An in-house data logger was chosen as the DIS data collection tool. The analysis tool used the data logs as input.

The analysis is composed of two executables. The first condenses raw binary scenario log files into a summary file which captures the significant scenario times and events. The second phase processes the summary file, based on a configuration file, and generates various statistics, including the aggregate MOE. The tool's configuration file specifies, for example, the weights associated with measures of effectiveness and vehicles that were not represented in the DIS traffic but should be reflected in the MOEs.

5. Experimental Results

It must be re-emphasized that the goal of this experiment was to test the feasibility of using virtual simulations for vehicle and weapon evaluation. The AAV results are useful to test and illustrate the analysis techniques. However, various simplifications were made and the AAV results, per se, should not be taken seriously.

5.1 Force Values

Entities are given three "force values" used to compute a side's force. A side's "force" is the sum of the forces for the side's entities, and the analysis tool's configuration file specifies the force represented by each entity. The force is a function of the entity's type and its health. Without more expertise, the assignments are somewhat arbitrary (in a full analysis, considerable care in this area would be necessary).

The force for an entity is denoted by three values: existence-force, mobility-force, and firepower-force. A healthy entity is given a force as the sum of these three. A destroyed entity has force zero. A healthy entity has at least its existence-force; it may also have its mobility or firepower force depending on its health. The DIS stream is rich enough to determine an entity's force.

Entities were partitioned into 3 groups; high-force (tanks, AAVs), medium-force (infantry teams), and low-force (trucks). With a few exceptions, the force values used are:

Force	Existence	Mobility	Firepower
High	3	4	4
Medium	2	3	3
Low	1	2	0

The low-force entities are given no firepower forces as any firepower they might carry is incidental.

5.2 Raw data

This is the raw MOE measures as generated by the analysis tool for scenario-1. The various quantities are computed as already described. All results are based on 80 runs.

MOE	μ	σ
N_VIC	0.7352	0.1275
N_OVN	0.7722	0.1261
N_RFR	0.7022	0.1476
N_LXV	0.5658	0.0971
N_LXF	0.4456	0.1322
N_LXP	0.3883	0.1518
N_CAS	0.3580	0.1599
N_SSTR	0.6211	0.1499
W_MOE	0.5736	0.0805
Variant-1		
MOE	μ	σ
N_VIC	0.7452	0.1196
N_OVN	0.7639	0.1316
N_RFR	0.7102	0.1473
N_LXV	0.5593	0.0969
N_LXF	0.4492	0.1175
N_LXP	0.3945	0.1295
N_CAS	0.3383	0.1346
N_SSTR	0.6348	0.1282
W_MOE	0.5744	0.0795
Variant-2		

5.3 Statistical Analysis

The individual MOEs are used to produce a combined MOE, as already described. In this table, each cell shows, for a given scenario/variant pair, the mean, and the standard deviation for the combination. The number of runs is all cases is 80.

	AAAV-X	AAAV-J
Scenario-1	$\mu=0.5736$ $\sigma=0.0805$	$\mu=0.5744$ $\sigma=0.0795$
Scenario-2	$\mu=0.6725$ $\sigma=0.0743$	$\mu=0.6722$ $\sigma=0.0704$

The samples are treated as independent samples drawn from a normal distribution. The object is to estimate the difference in the means for the variants for each scenario.

Symbolic representations in this analysis use subscripts to denote scenarios and variants. For scenario-s, variant-v, the mean, standard deviation, and number of trials for an experiment are denoted μ_{sv} , σ_{sv} , and N_{sv} respectively.

The maximum likelihood estimate for the difference in means for the scenario-s MOE is $\mu_{sx} - \mu_{sj}$. A positive value indicates an advantage for the AAAV-X, a negative value indicates an advantage for AAAV-J. This raw figure cannot be taken seriously in light of the variation in sampling (as indicated by the standard deviation).

A pooled sample variance for $\mu_{sx} - \mu_{sj}$, denoted σ_s^2 , is used for the overall deviation:

$$\sigma_s^2 = \frac{(N_{sx} - 1)\sigma_{sx}^2 + (N_{sj} - 1)\sigma_{sj}^2}{N_{sx} + N_{sj} - 2}$$

Because $N_{sx} = N_{sj} (= 80)$,

$$\sigma_s^2 = (\sigma_{sx}^2 + \sigma_{sj}^2)/2$$

(the combined variance is the mean of the sampled variances).

Using this, a confidence interval for $\mu_{ax} - \mu_{aj}$ is easily computed. Enough data is available for all of the experiments ($N_{sx} + N_{sj} - 2 > 30$) to use a large-sample confidence interval.

$$(\mu_{sx} - \mu_{sj}) \pm (Z_{\alpha/2})(\sigma_s) \sqrt{\frac{1}{N_{sx}} + \frac{1}{N_{sj}}}$$

For a 95% confidence interval, $Z_{\alpha/2} = 1.960$. $N_{sx} = N_{sj} = 80$. Thus, our interval, to four significant digits, reduces to:

$$(\mu_{sx} - \mu_{sj}) \pm (0.3099)\sigma_s$$

5.3.1 Analysis for Scenario-1

From the above equations, the confidence interval for $(\mu_{1x} - \mu_{1j})$ is:

$$(0.5736 - 0.5744) \pm (0.3099)(0.0800)$$

or

$$(-0.0256, 0.0240)$$

Since this interval includes zero, no conclusion can be drawn as to which variant is superior based on this data. With additional information (increasing N_{sx} and N_{sj}) the confidence interval can be reduced to the point where a conclusion can be reached.

5.4 Hypothesis Testing

A hypothesis test may also be used. There is a duality of confidence intervals and hypothesis testing, and either may be applied to the case at hand. Earlier computations, such as the pooled sample variances, are used here.

The obvious hypothesis is that the variants have different MOEs, but the likelihood of a type II error (β) if that is used as the null hypothesis is unclear. So, as is often done, we reverse roles use $\mu_{ax} = \mu_{aj}$ as the null hypothesis. This way we can know the likelihood of incorrectly deciding that $\mu_{ax} \neq \mu_{aj}$ when they are actually equal (this is a type I error and we can control its value, α).

$$H_0: \mu_{sx} - \mu_{sj} = 0$$

$$H_a: \mu_{sx} - \mu_{sj} \neq 0$$

Our sample sizes are large enough for approximate normality of our sample means to hold. Our test statistic, Z , is

$$Z_s = \frac{\mu_{sx} - \mu_{sj}}{\sigma_s \sqrt{\frac{1}{N_{sx}} + \frac{1}{N_{sj}}}}$$

For both scenarios, the radical reduces to approximately 0.158.

We have a two sided alternative, and so $|Z|$ will have to exceed $Z(\alpha/2)$ for H_0 to be rejected. Seeking 95% confidence, $\alpha=0.05$ and $Z(\alpha/2) = 1.960$.

For scenario 1,

$$Z = 10.5736 - 0.5744 / ((0.205)(0.158)) \approx 0.025$$

not within the rejection region. For scenario 2,

$$Z = 10.6725 - 0.6722 / ((0.07235)(0.158)) \approx 0.026$$

also outside of the rejection region.

Thus, in both cases, H_0 cannot be rejected, so we cannot conclude $\mu_{sx} - \mu_{sj} \neq 0$. This is consistent with the confidence interval result.

5.5 Additional Trials

More trials will increase N_{sx} and N_{sj} , and so will reduce the size of the confidence intervals. With enough trials, we should be able to distinguish between the variants, if they are different. The results are far too close to trust this additional analysis; it is included here for demonstration purposes only.

For scenario-1, the means differ by (only) -0.0008. Assuming we continue to use the same number of trials for each variant (call this N_s), for the confidence interval to exclude zero:

$$Z_{\alpha/2} \sigma_1 \sqrt{\frac{2}{N_s}} < 0.0008$$

or, for scenario-1 and a 95% confidence interval:

$$(1.96)(0.08) \frac{1.414}{\sqrt{N_s}} < 0.0008$$

or

$$277.1 < \sqrt{N_s}$$

which indicates that over 76,000 more runs are needed for each variant (for a total of over 150,000 more trials; this would take over 11 weeks of continuous logging).

5.6 Variant Conclusions

Based on these experiments, there is no difference between the variants. This conclusion says more about simplifications in the experiments than the variants themselves.

5.6.1 Problematic Simplifications

Among the simplifications that may have lead to the negative conclusion:

- AAV-X and AAV-J are **identical** except that 1/3 of the AAV-J's carry Javelin ammo. Any differences must come about from this one difference.
- Both variants used the same tactics (in fact, used the same scenario). A full blown test should use tactics appropriate to the available weapons.
- Only two scenarios were tried. A richer scenario mix may uncover important differences even for these variants.
- The underlying CGF system forced many compromises, some of which may have hidden variant differences.

6. Overview of Experimental Problems

Several flaws are obvious when the methods applied are compared with the recommendations. These flaws reflect the nature of the experiments and a reduced emphasis on attempting to actually reach conclusions regarding the AAV variants. The experimental work did not require a greater emphasis in these areas to yield the *techniques* sought. Highlights of flaws in applying the recommendations include:

- the scenarios were not customized for the AAV variant being tested,
- the scenario selection list was far too short,
- the MOEs reflect best guesses (they are not in any way validated),
- the AAV implementation lacks details (such as armor modifications), and
- vehicle and weapon data is not accurate (as far as practical it does reflect supplied information).

6.1 Problems with the CGF

Other experimenters are likely to encounter problems of the sort encountered with ModSAF. All such systems are likely to be inadequate in some areas. To give a notion of what to expect, and to clarify the nature of the experimental work, the most striking problems encountered are mentioned here; there were many more.

Variable Platoons: The vehicle count in ModSAF platoons is tied to the vehicle type (T72 platoons uni-

formly consist of three vehicles). AAV platoons needed to be of variable size.

Entity Counts: The size of the execution matrix⁴, which is unaccounted for in ModSAF's maximum entity benchmark, appears to have a dramatic impact on the maximum vehicle count. This dilemma placed constraints on scenario design; as scenarios became more complicated (manifested as larger execution matrices) the number of vehicles that could be modeled dropped.

Performance: ModSAF monitors its performance and announces overloading as "gassing." When the system is past the gassing threshold, behaviors break down. Gassing was a serious problem throughout this project.

Sound Modeling: No sound modeling exists in ModSAF but the experimental scenarios required control measure triggers on the sound of approaching vehicles and weapons fire. To mimic this, a line of sight requirement was relaxed for a specialized ModSAF task transition mechanism.

Radio Control Measures: Task transition based on inter-vehicle communication is a realistic, and necessary, missing feature. Without knowledge of task completion, timers and transition lines were used to trigger transitions although this technique is, at best, imprecise.

Indefinite Holds: An observed unit crossing a control measure before the observing unit is in position causes an indefinite hold on the current task of the observing unit.

Infantry Mounting Restrictions: An infantry team that is intended to mount must be created in tandem with the vehicle to be mounted. The infantry team is then restricted to mounting its partner vehicle (ModSAF does not allow the substitution of one Personnel Carrier for another).

Mounted Infantry Immortality: ModSAF does not kill mounted infantry when the vehicle they are on experiences a catastrophic kill; they simply are not allowed to dismount. This would yield underestimates of the casualties in the MOEs. Workarounds were implemented in the analysis tool.

⁴ ModSAF tasks are assigned to entities (including units) via an "execution matrix." This is a sequential list of entity's tasks and task transition specifications.

Close Air Support: No reaction is available to call for close air support. To explain the lack of air support in the scenarios, bad weather conditions were hypothesized.

Javelin Targeting: On those occasions that a AAV fires a second Javelin while one is in flight, the second always uses the same target as the first (this is very unrealistic).

Poor Reload Behaviors: reload is immediate, leaving the AAVs vulnerable for about one minute during reload. Reload should be done under cover.

Locked Target Priorities: a Javelin equipped AAV should not seek tanks to kill but it should defend itself. As ModSAF stands, tanks are targets (on the priority list) or not (off the priority list).

Weapon Targets: appropriate weapons selection is not available; in order to have AAV-J attack tanks at all, the tanks had to be on the priority list. However, in that case AAVs attacked whether they had a Javelin on board or not.

Nominal Entities: In some cases ModSAF developers have made compromises by modeling one entity by simply mapping it to another (e.g., the underlying Javelin model is a TOW missile model). The names used ("Javelin") can give false confidence in results.

PO Database: multi-station ModSAF runs experienced packet losses under PO database⁵ bursts. Such losses jeopardize experimental results and were avoided by performing the experiments on a "Pocket Simulator" (a single station), which eliminated the need for PO network traffic. To accomplish this, the number of entities had to be minimized.

Shut Down: During scenario shut-down, DIS traffic appears to be suspended, then, after a considerable delay (sometimes over a minute), more DIS traffic is transmitted. The analysis tool uses inter-traffic gaps to recognize scenario breaks and so this was a real problem. Adjustments to the tool (ignoring "short" scenarios) compensated for the problem.

Hasty Occupy & Attack by Fire: While executing either Hasty Occupy Position or Attack by Fire, entities try to close up ranks when a vehicle is destroyed along

⁵ ModSAF uses version-dependent, non-standardized, "Persistent Object" database network traffic (possibly too free-wheeling to be properly called a "protocol") to keep its stations synchronized.

the occupied battle line. This stops fire until the ranks close, causing some vehicles to expose their side armor while the ranks close (and is generally unrealistic).

Force Confrontation: Opposing forces often begin a battle with heavy firing, followed by a period with a lone surviving vehicle actively fighting on one side and one or more (apparently) idle vehicles on the opposing side. Eventually, the isolated vehicle is killed, but not before it had time to inflict damage.

Fire Permission: Fire permission cannot be changed from task to task, although it is automatically changed during reactions. Positions and routes had to be selected to avoid LOS and so avoid unintended firing.

Command Line Options: ModSAF crashes when the "sourcefile" option is specified in conjunction with the "nogui"⁶ option. The sourcefile option was essential to automate the experiments. The system was run with a GUI in spite of the performance costs.

Scenario Editing: There is no way to insert new tasks into the execution matrix other than at the end of the matrix. This resulted in terrific overhead for scenario generation. Seemingly small changes, requiring insertion of a new task, required a complete re-build of the scenario. Old scenarios could not be loaded into new AAV versions of ModSAF when a change in the AAV PO Database definition was introduced.

7. Conclusion

CGF systems can be used for evaluating future systems but it is a complex process. A CGF must be altered to support the system to be tested, a variety of appropriate scenarios must be developed and implemented, MOEs are needed for the scenarios and should be instantiated in software, and enough experimental runs are needed to determine statistically significant results.

Each step requires personnel with special expertise. For example, it is unlikely that the people who carry out software development will also have the knowledge to develop appropriate scenarios for the experiment.

It is recommended that the MOEs be based on network traffic (e.g., DIS) as this breaks a key binding between result analysis tools and the CGF work (development of analysis software and scenarios support work can be

done in parallel, and different versions of each may be developed without impacting the other).

An experimental use of the technique, in spite of severe simplifications (making the results of the system analysis valueless), turned out to be a large project which uncovered many problems in the CGF system used. To carry out the experiment to the point where the results would be trustworthy, using the same CGF system, would be a more complex and difficult problem. Given a mature, verified, and validated CGF appropriate for the system under test the experimental development's complexity could be reduced tremendously.

8. Author's Biographies

Michael A. Craft is a Senior Computer Scientist at the Institute for Simulation and Training. Mr. Craft has an M.S. in Computer Science and an M.S. in Mathematics. His major interests include computer protocols, software engineering, and systems development.

Clark R. Karr is the Program Manager of the Computer Generated Forces projects at the Institute for Simulation and Training. Mr. Karr has a Master of Science degree in Computer Science. His research interests are in the areas of Artificial Intelligence and Computer Generated Forces.

9. References

Courtemanche, A. J., and Ceranowicz, A. (1995). "ModSAF Development Status," Proceedings on the Fifth Conference on Computer Generated Forces and Behavioral Representation, Orlando, FL, Institute for Simulation and Training, May 9-11 1995. pp. 3 - 13.

Craft, Michael A., Kraus, Matthew K., Mullally, Daniel E., Adkins, Michael K., Albright, Robert L, Nida, Jonathan C., Napravnik, Lee J. (1995) "AAAV: Demonstrating the Feasibility of Using Virtual Simulation for Test and Evaluation," Technical Report IST-CR-95-32, Institute for Simulation and Training, University of Central Florida, 90 pages.

⁶ This option disables ModSAF's Graphical User Interface freeing up machine resources. With "nogui" it is possible to support more complex scenarios.

Use of ModSAF in Development of an Automated Training Analysis and Feedback System

Theodore Metzler and John Nordyke
LB&M Associates, Inc.
211 SW A Ave
Lawton, OK 73501-4051
metzlert@lbm.com nordykej@lbm.com

1. Abstract

This paper describes a use of simulations with Computer Generated Forces (CGF) in which ModSAF supported the testing of rule-based Artificial Intelligence (AI) modules in an Automated Training Analysis and Feedback System (ATAFS). A brief overview of the ATAFS tool is presented with accompanying figures to explain the developmental context of the testing. Testing methodology is then described and illustrated with a number of specific examples of how ModSAF was used. Results of testing are reported and evaluated, identifying certain advantages offered by the CGF tool for applications of this kind. Synergy of interacting CGF simulation and AI reasoning demonstrated in this testing may also benefit other development efforts—a possibility addressed in our concluding review of further ModSAF applications suggested by the work we describe.

2. System Background

ATAFS constitutes one application of a more general technology-based capability to “eavesdrop” on selected data streams, collecting, analyzing and displaying information. The ATAFS workstation is an expert system-based after action review (AAR) tool that automatically produces AAR aids for simulation networking (SIMNET) exercises, with potential to support other virtual, constructive and live simulations. The AAR aids generated by ATAFS include discussion points, animated plan views of the battlefield, displays showing shotlines and artillery impacts with traces of unit movement, graphs, tables and replays of voice communications produced synchronously with top-down views of the player unit’s activities. A composite illustration of some of these features is shown in Figure 1.

Operated interactively by an Observer/Controller (O/C), ATAFS records and monitors SIMNET messages, using its rule-based AI components to identify automatically the occurrence of certain battlefield events. ATAFS prepares AAR aids without assistance from the O/C for those tactical events the system is able to recognize. For example, ATAFS can detect direct and indirect fires, vehicle kills and vehicles crossing control measures such as the Line of Departure (LD). Control measures of this kind are graphically designated for the system on a digitizing tablet by the O/C, according to operating instructions in the ATAFS user’s manual (LB&M 1996). Simulation events that mark the start or end of an AAR aid are recognized by ATAFS software in terms of “triggers.” A sample of these events and triggers is shown in Figure 2. Comparison of Figure 2 with Figure 1 will help clarify the foregoing description for the illustrative battlefield event, “Movement from LD to First Enemy Contact.” AI modules of ATAFS, implemented in CLIPS v6.0, monitor and interpret SIMNET messages to detect the “triggers” using rules of the type illustrated in Figure 3.

To ensure the rule sets correctly and reliably recognize selected battlefield events in the network traffic, we employed ModSAF v1.2.2 as a testing tool, generating developer-controlled network activity that the AI components of ATAFS could monitor and interpret.

3. ModSAF Application

A prototype ATAFS workstation played a passive role in the testing configuration, eavesdropping on an ethernet network carrying SIMNET messages. The messages were generated by a ModSAF simulator, which

developers successively set up to produce selected test scenarios. Unlike some other ModSAF development applications involving interaction of prototype AI objects with CGF entities (Laird 1995, Tambe 1995), our tests challenged the rule-based modules of ATAFS only to "observe" and correctly interpret the

assembled them allowed detection of deficiencies in rule sets. Figure 4 depicts the simple configuration used for this iterative procedure of testing and rule refinement.

Battlefield scenarios employed in the procedure involved "Force on Force" activity of CGF

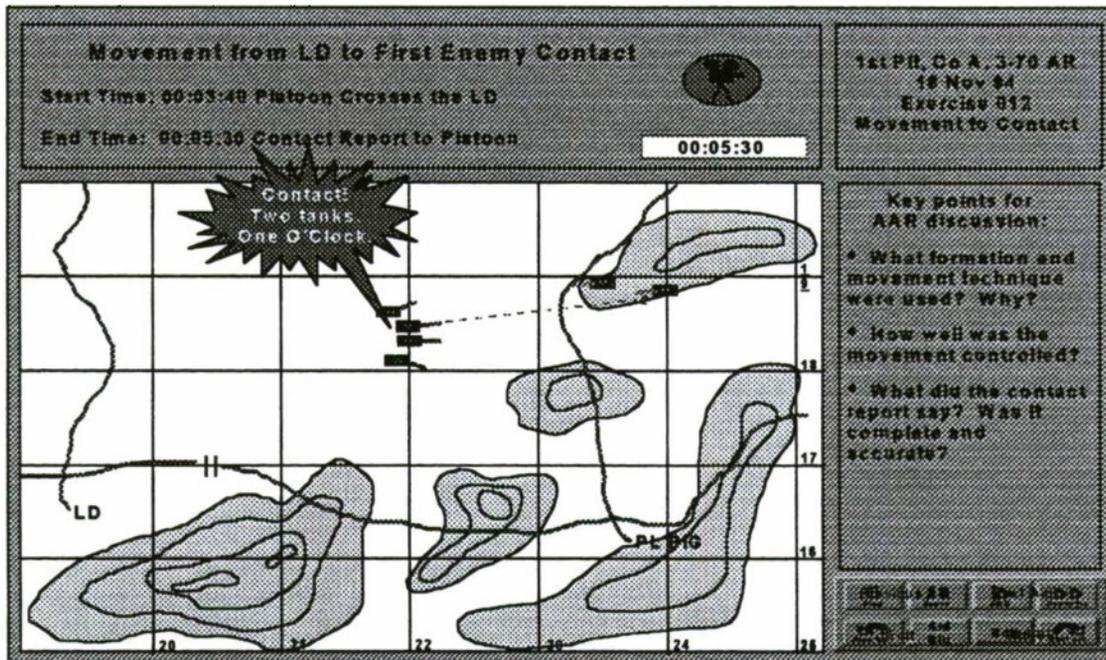


Figure 1. ATAFS Features

unfolding CGF scenarios. For each test scenario members of the development team functioned as O/Cs, operating the ATAFS workstation to capture desired AAR aids. Subsequent comparison of the AAR aids actually produced with the test scenarios from which ATAFS

entities--for example, pitting one BLUFOR armor platoon against one or more OPFOR armor and/or motorized infantry platoons. Defensive, offensive and tactical road march scenarios were simulated by ModSAF in the testing.

Aid Series: 2		
Exercise Phase: Movement from LD to First Enemy Contact		
Aid Number, Title and Type	Possible Starting Events and Triggers	Possible Ending Events and Triggers
2.1 Platoon Crosses LD (PVA)	<p>EVENT: 1st vehicle crosses LD.</p> <p>TRIGGER: Line Trigger used so ATAFS can sense the unit's crossing of the LD.</p>	<p>EVENT: Contact report sent.</p> <p>TRIGGER: OC acuates Contact Report prompt.</p> <p>EVENT: BLUFOR fires first direct fire round.</p> <p>TRIGGER: ATAFS senses first BLUFOR round fired using the Firing Trigger.</p> <p>EVENT: OPFOR fires first direct fire round.</p> <p>TRIGGER: ATAFS senses first OPFOR round fired using the Firing Trigger.</p>

Figure 2. Events and Triggers

4. Results

The testing with ModSAF proved to be a valuable developmental procedure, disclosing a number of needed corrections in the prototype rule sets of ATAFS. For example, it was discovered that when vehicles entered an objective and remained there, the rules initially generated spurious training aids indicating exit of the vehicles from the objective. Repaired versions of these defective rules were tested with

check for each test run to ensure that all rule triggering events were thoroughly exercised. In general, the flexible ModSAF support for creating a wide range of custom scenarios permitted our analysts to confirm rule sets that work (versus "should work").

5. Future Directions

Enhancements of the ATAFS workstation are expected to include an "authoring tool," allowing users who are not programmers to extend the set of AAR aids ATAFS produces. Extensions created with this tool will also occasionally need to be tested. To satisfy this need, we may reasonably consider a lesson from the developmental experience just described and make a subset of ModSAF capability available for use with the authoring tool.

In actual simulation exercises, we have also observed that the ATAFS workstation is often operated concurrently with a separate workstation from which ModSAF generation of CGF entities is controlled. This practice introduces redundancy, since ATAFS and ModSAF share some common representations for input of control measures, overlays, etc. Accordingly, future versions of ATAFS may be more immediately connected with ModSAF, allowing one operator at a single workstation to direct CGF parts of a simulation while building AAR aids with ATAFS.

Finally, it may be possible for ModSAF to assist the development of AAR systems such as ATAFS in ways somewhat different from the direct testing we have described. In particular, work previously reported at this conference regarding the automated knowledge acquisition system known as "Captain" (Hieb 1995, Hille 1994) suggests an interesting potential linkage of ATAFS, Captain and ModSAF. As a tool for building intelligent (CGF) command agents, Captain may offer useful resources to development of future ATAFS workstations, since much of the situation awareness and reasoning required of the battlefield commanders modeled by Captain is also employed by O/Cs in production of AAR aids. In turn, ModSAF supports Captain's adaptive modeling of such behavior in several interactive learning modes. Hence, ModSAF may continue to benefit future ATAFS development through a

training role as well as the more immediate testing role reported in this paper.

6. Acknowledgment

Technical contributions to preparation of this paper by Joseph Kelly, LB&M Systems Analyst, are greatly appreciated.

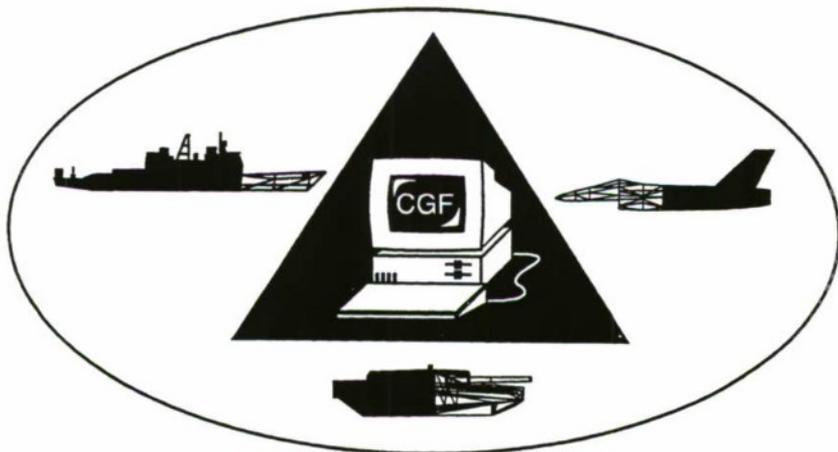
7. References

- Hieb, M.R.; Tecuci, J.; Pullen, J.M.; Ceranowicz, A.; and Hille, D. "A Methodology and Tool for Constructing Adaptive Command Agents for Computer Generated Forces." In Proceedings of the Fifth Conference on Computer Generated Forces and Behavioral Representation, pp. 135-146. Orlando, Florida: Institute for Simulation and Training, 1995.
- Hille, D.; Hieb, M.R.; and Tecuci, G. "Captain: Building Agents that Plan and Learn." In Proceedings of the Fourth Conference on Computer Generated Forces and Behavioral Representation. Orlando, Florida: Institute for Simulation and Training, 1994.
- Laird, John E.; Johnson, W. Lewis; Jones, Randolph M.; Koss, Frank; Lehman, Jill F.; Neilson, Paul E.; Rosenbloom, Paul S.; Rubinoff, Robert; Schwamb, Karl; Tambe, Milind; Van Dyke, Julie; van Lent, Michael; and Wray III, Robert E. "Simulated Intelligent Forces For Air: The Soar/IFOR Project 1995." In Proceedings of the Fifth Conference on Computer Generated Forces and Behavioral Representation, pp. 27-36. Orlando, Florida: Institute for Simulation and Training, 1995.
- LB&M Associates, Inc. Software User's Manual for the Automated Training Analysis and Feedback System. Lawton, Oklahoma: LB&M Associates, 1996.
- Tambe, Milind; Schwamb, Karl; and Rosenbloom, Paul S. "Building Intelligent Pilots for Simulated Rotary Wing Aircraft." In Proceedings of the Fifth Conference on Computer Generated Forces and Behavioral Representation, pp. 39-44. Orlando, Florida: Institute for Simulation and Training, 1995.

8. Authors' Biographies

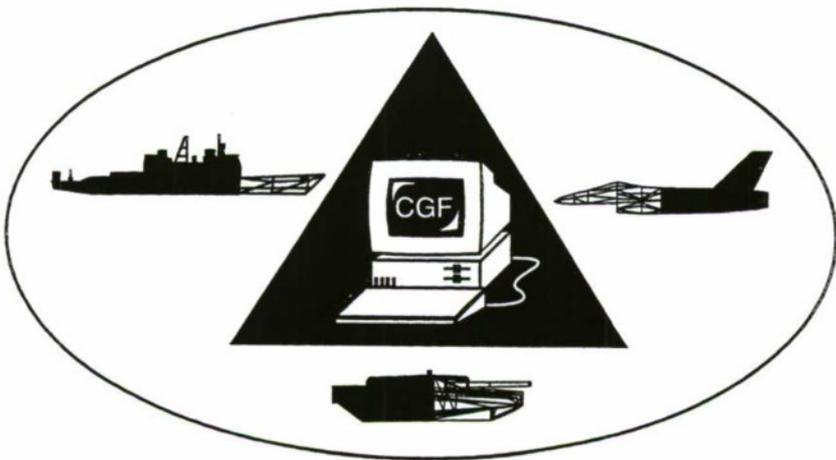
Theodore Metzler is a Senior Systems Engineer at LB&M Associates, Inc. Mr. Metzler has an M.S. degree in Computer and Communication Sciences and a Ph.D. in Philosophy. His research interests are in the areas of Hybrid Artificial Intelligence and Artificial Neural Networks.

John Nordyke is a Systems Analyst at LB&M Associates, Inc. He has a B.S. degree in Mathematics and an M.A. in Management and Human Relations. His research interests are in the areas of Graphical User Interfaces, Artificial Intelligence/Expert Systems and Simulation Training Systems.



Session 3a: Behavior Representation

Kraus, UCF/IST
McKenzie, SAIC
Balzer, USC/ISI
Rajput, UCF/IST



CCTT SAF and ModSAF Behavior Integration Techniques

Matthew K. Kraus, Derrick J. Franceschini, Tracy R. Tolley, Lee J. Napravnik,
Daniel E. Mullally and Robert W. Franceschini
Institute for Simulation and Training
3280 Progress Drive, Orlando, FL 32826
mkraus@ist.ucf.edu

1. Abstract

As CGF systems have matured over the last ten years, they have been applied to solving increasingly difficult problems. The analysis community would like to use CGF systems that portray battlefield effects at the individual vehicle level. For credible analysis, it is desirable that CGF behaviors be derived from military doctrine and be traceable back to that doctrine.

The CCTT SAF program has undertaken a large knowledge engineering effort to produce realistic CGF behaviors. Part of this effort transforms military doctrine into Combat Instruction Sets (CISs), a natural language description of tactical behavior. Because they are based on CISs, CCTT SAF behaviors are traceable to military doctrine.

This project's goal is to research methods of incorporating these CISs into ModSAF¹. This has consisted of several phases of work. The first phase was to research CISs to understand their structure and complexity. Next, IST enumerated differences between CCTT SAF and ModSAF that could affect CIS integration. Then CCTT SAF code and corresponding documents were used to provide more insight into the CCTT SAF environment and its implementation of selected CISs. Lastly, two prototype CISs were implemented in ModSAF. This proof of concept successfully illustrated the feasibility of incorporating traceable behaviors in ModSAF.

2. Background

CIS behaviors bring obvious realism and credibility benefits to the research and training communities. Reusing CCTT SAF technology leverages the investment the US government has made in simulation and training. IST leveraged the CCTT SAF software development effort to help implement CISs in ModSAF.

¹ All unqualified references to ModSAF are to version 2.0.

IST's research first focused on several behavior integration issues. IST researched CISs to understand their structure and complexity. IST then found differences between CCTT SAF and ModSAF that could influence behavior interoperability. Finally IST developed a process for implementing CISs in ModSAF. Two prototype CISs were implemented in ModSAF as a proof of concept.

3. CIS General Information

IST studied Combat Instruction Sets (CISs) before implementing a prototype behavior. CISs are designed to allow someone unfamiliar with military doctrine to understand the actions taken by a unit executing a behavior. CCTT SAF uses CIS descriptions found in the CATT-Task Database to produce doctrinally correct actions for each behavior. The CATT-Task database combines training data from task manuals, soldier manuals, subject matter experts, and training studies into one source (Wright 1994).

3.1 Sources

BLUFOR² CISs are derived from U.S. Army Training and Evaluation Program (ARTEP) Mission Training Plans (MTPs). Because they were derived from U.S. doctrine, BLUFOR CISs contain more detailed information than their OPFOR counterparts. They are denoted by Bxxxx where the number xxxx corresponds to the unit type.

3.2 Elements of a CIS

A CIS contains a behavior description, a sequence of actions to be taken in the behavior, initial conditions, input data, terminating conditions, and situational interrupts. In the Actions to be Taken section of the CIS, each action in a BLUFOR behavior is grouped into *move*, *shoot*, *observe*, or *communicate* based on the nature of the action. OPFOR CISs only list the actions in their order of execution. Initial conditions detail necessary information for the behavior to

² BLUFOR refers to U.S. Forces and OPFOR refers to Opposing Forces.

execute. Terminating conditions outline reasons for behavior completion and what actions to take when the behavior has finished. Situational interrupts describe reactive behaviors that could interrupt the behavior.

3.3 Complexity

One measure of implementation complexity is the relationship between CISs. In general, IST determined that each CIS requires other CISs as part of its specification. Therefore, the complete implementation of one CIS requires implementation of many other CISs. For the purposes of this project, IST has limited the problem by substituting existing ModSAF behaviors for the supporting CISs.

4. Differences Between CCTT SAF and ModSAF

IST investigated the feasibility of implementing CISs in ModSAF. Because CCTT SAF uses CISs for their behaviors, adding CISs to ModSAF would improve the interoperability between these two CGF systems. Differences between the underlying architecture of these two systems could impact the behavioral interoperability between CCTT SAF and ModSAF. Over one hundred issues were found that could impact behavior interoperability. Due to space limitations, only a few of the issues are discussed here. These issues can be grouped into the following categories:

- Command and Control Hierarchy
- CGF Services
- Task Management
- Reactive Behaviors
- CCTT SAF FSMs vs. ModSAF AAFSMs
- Environment and Terrain
- Code Sharing
- Crew Level Behaviors

This section examines some differences between CCTT SAF and ModSAF that affects behavior integration.

4.1 Command and Control Hierarchy

CCTT SAF uses a “ghost” controller associated with a platoon as the platoon leader. The ghost controller is associated with a simulated entity, but it is not a simulated entity itself: it does not have a physical model, take damage, or interact with other entities in the battlefield. When its associated vehicle is destroyed, the ghost controller is assigned to another vehicle. All information gathered is maintained and

the task continues (Marshall 1996). A platoon leader in ModSAF is assigned to a particular vehicle in the platoon. If this vehicle is destroyed, ModSAF restarts the entire task with the role of platoon leader assigned to a new vehicle. All previous knowledge that this task has acquired since the task was initialized is lost (Rajput and Karr 1995). Note that neither system completely reflects what occurs in real life. In the real world, the new platoon leader should have some of the knowledge that the previous one had, but it would take time to assimilate information that the previous platoon leader gathered.

The two systems also handle communication between subordinates and commanders differently. In CCTT SAF, superiors send orders to subordinates in the SAF Entity Object Database (SEOD) (Horan 1994). ModSAF does not have the concept of orders, rather it has superiors start tasks for subordinates using procedure calls.

4.2 CGF Services

Even with identical behavioral logic and supporting data at a given echelon level, behaviors can produce different results because of dissimilar underlying services. Some examples of these underlying CGF Services are terrain reasoning, weapon system modeling (e.g., assessing, enemy detection, and targeting), physical modeling (e.g., hull and turret), and sensor modeling (e.g., visual, infrared, radar).

Routing and searching for covered and concealed positions requires terrain analysis. While CCTT SAF’s dynamic and static obstacle avoidance algorithms are based on existing ModSAF algorithms, there are differences. For example, IDA* was used in CCTT SAF for planning road routes instead of A*, which is used in ModSAF (Campbell et al. 1995). Further research is necessary to determine the extent of the differences between the CGFs in other CGF services (e.g., other areas of terrain reasoning, weapons system modeling, physical modeling, and sensor modeling).

Differences in these underlying behaviors could be perceived as the distinct methods individual soldiers would use were they executing one of these tasks (e.g., two drivers may choose two different routes through the same forest to reach the same destination).

4.3 Task Management

Execution of a scenario consists of coordinating and executing a series of behaviors. Given that a set of behaviors from two CGF systems are identical, differences in task management can affect overall behavior and scenario outcome. Currently, IST lacks information about CCTT SAF's task management methodology. Because of this, only general issues that affect behavior interoperability of two CGF systems will be addressed here.

4.3.1 Task Scheduling (Priorities and Hierarchy)

For two CGFs to have interoperable behaviors, their task scheduling mechanisms must be similar. CCTT SAF and ModSAF use ring queues to manage time based and priority based task scheduling. Tasks in CGF Services, among other things, are grouped into schedule rings based on the number of times per second that they need to be executed, e.g., all tasks that need to be executed 15 times per second are assigned to the 67 millisecond ring. As vehicles and units are created, several CGF Service tasks (e.g., routing, assessing) associated with vehicles or units are initialized and assigned to their proper rings. The rate at which all CGF Service tasks are executed in both CGF systems should be similar for behavioral interoperability between the CGF systems.

4.3.2 Task Execution (Ticking and Task Transitions)

Execution of a scenario involves the coordination and execution of a sequence of behaviors. Transitions between behaviors can be automatic, triggered by a Control Measure, or require operator intervention. There are ModSAF behaviors that do not automatically transition to a subsequent behavior when they complete (e.g., Hasty Occupy Position), while other behaviors do automatically transition (e.g., Road March). Knowledge of CCTT SAF's handling of task transitions would allow more insight into how interoperable a sequence of behaviors could be in comparison to a similar sequence in ModSAF.

A 'wrapper' is code that executes before or after a user specified behavior. Any wrapper placed around behaviors in CCTT SAF must be identical to those in ModSAF for a sequence of behaviors to act similarly. ModSAF requires a preparatory task, a preliminary task executed before the actual behavior, for each behavior (HALT is most commonly used). The advantage to using a preparatory task is that each behavior starts from a known condition. The disadvantage is that certain sequences of tasks exhibit odd behavior. For example, sequential move tasks will not keep a vehicle in continuous movement. The

second move (as well as the first) starts from a halted state, i.e., the vehicles stop between each move task. If CCTT SAF handles this differently, a behavior in CCTT SAF would act differently than its equivalent in ModSAF.

The *tick rate* is the maximum frequency that a task is executed. As a simulation becomes busy, the time required to execute tasks in a ring queue can exceed the assigned time for the ring, compromising simulation fidelity and the "real-timeness" of the system (Smith and Swarts 1990). Symptoms of a busy simulation are movement and behavior degradation (Vrablik and Richardson 1994). Behavioral inconsistencies may arise because CGF system A may assign more items to a given ring than system B. As the number of items on a ring increases, it will become more difficult for those items scheduled on a ring to complete on time.

4.4 Reactive Behaviors

ModSAF behaviors do not correspond to CIS definitions. Consequently, the full implementation of a CIS will require the addition of CIS reactive behaviors in ModSAF.

For some behaviors, CCTT SAF incorporates the code for a reactive task into the code for a non-reactive task. For example, in the OPFOR Assault an Enemy Position, CCTT SAF will execute code to breach an obstacle inside the Assault an Enemy Position task instead of calling a standard Breach Obstacle task. ModSAF transitions to a reactive task by starting the appropriate task. The limitation of CCTT SAF's approach is that every behavior needing Breach Obstacle must incorporate all the code for Breach Obstacle again. An advantage to doing this is that Breach Obstacle could be tailored to a specific behavior, i.e. an assault Breach Obstacle may need to be different from a traveling Breach Obstacle. Although this presents no behavioral interoperability difficulties, having redundant code presents software maintenance problems.

Reactive Behavior	ModSAF (2.0)	CCTT SAF
Ambush (OPFOR)	No	Yes
Consolidate and Reorganize	No	Yes
Execute Appropriate Action Drill	No	Yes
Execute Contact Drill	Yes ³	Yes
React to Indirect Fire	Yes ³	Yes
React to Terrain	Yes ³	Yes
Recon Drills (OPFOR)	No	Yes
Take Actions At Obstacle	No	Yes
Take Active Air Defense while Moving/Stationary	Yes ³	Yes

Table 1 – Reactive Behaviors

Table 1 illustrates some sample reactive behaviors and whether they are supported in the two CGF systems.

4.5 CCTT SAF FSM and ModSAF AAFSM

Finite State Machines (FSMs) are often used to describe behaviors. An FSM consists of states and transitions. Each state in an FSM corresponds to either a function or a low-level FSM. Transition conditions associated with each state make up the criteria for entering another state (Smith and Petty 1992). Note that CCTT SAF and ModSAF implement different variations of FSM structures. CCTT SAF and ModSAF have several implementation differences in their FSM structures that could affect either the way that a behavior is executed or the ease of implementing a behavior. These differences are exhibited in the implementation language, preprocessing steps, and task parameter changes.

CCTT SAF's FSMs, written in Ada, contain all the details that link FSMs into the CCTT SAF. On the other hand, ModSAF FSMs, written in C-like syntax (Asynchronous Augmented Finite State Machine or AAFSM format), must go through a preprocessing stage before they become C code. This abstracts out details of FSM linkage to the system and thereby accelerates behavior development.

³ Modifications necessary for CCTT SAF compatibility

CCTT SAF does not respond to changes in an input to a behavior once the task is running, restricting the operator from responding to changes in orders. ModSAF's AAFSM format explicitly allows this change of parameters. For example, if a frag order updates a phase line's position, then the operator may move it and the task will respond to this change, instead of having the operator reissue the task with the new parameters.

4.6 Environment and Terrain

Environmental elements may affect behaviors (e.g., smoke, rain, fog, snow, etc.). These will cause a degradation in various behaviors due to reduced sensory input, reduced traction, and reduced trafficability. A CGF system's behavior is limited by the fidelity of its terrain. High fidelity terrain provides more covered and concealed positions and objects for entities to interact with (e.g. log cribs, tank ditches, DI berms, etc.) than lower fidelity terrain.

Because they represent the operating environment for entities, environment and terrain differences can play an important role in affecting unit behaviors. In general, CCTT SAF has more detailed environment features and terrain than ModSAF.

4.6.1 Environment

Deviations in support for environmental factors will cause two CGFs to act and react differently. This affects behaviors by increasing sensor degradation and reducing trafficability.

CCTT SAF and ModSAF support sensor degradation due to rain, fog, and haze. Trafficability in CCTT SAF is reduced due to rain soaking the ground (if the rain floods an area, vehicles will route around it, and traction is reduced on rained soaked terrain). ModSAF does not currently support rain soaked terrain. By ModSAF ignoring the impact of weather on routing (e.g., avoiding muddy terrain), its behaviors will not be interoperable with CCTT SAF.

4.6.2 Terrain

The terrain database representation in CCTT SAF has varying polygonal facets of 60, 120, and 240 meters. This is multi-level terrain, providing trafficability both over a bridge and through the water beneath it (Pope et al. 1995). ModSAF's terrain database can have different polygonal facets and there are no multi-level terrain features (Braudaway et al. 1995). Forests in CCTT SAF are represented as tree aggregates, each of which can be broken up into its respective trees (Pope et al. 1995). Forests are represented as canopies in ModSAF with no information about individual trees contained inside the forest (Braudaway et al. 1995). CCTT SAF's terrain database contains up to 10,000 destroyable 3D features. The terrain also contains many relocatable objects (those that can be moved around the terrain) in the form of log cribs, tank ditches, DI berms, etc. (Pope et al. 1995). There are few destroyable or relocatable objects in ModSAF (e.g., ModSAF has AVLB vehicles) (Braudaway et al. 1995).

CCTT SAF's varying polygonal facets and support of closer grid posts allow for more accurate terrain representation. This terrain format allows vehicles in high traffic areas to have more terrain objects (trees, buildings, water, etc.) to interact with than ModSAF's terrain format provides. The CCTT SAF's use of tree aggregates allows an entity or unit routing through a forest to be able to use the trees for concealed positions and to have its route affected by more obstacles (in the form of trees). Relocatable and destroyable objects provide a more realistic environment for the entity. An entity may take advantage of a relocatable object for concealed positions or face obstacles because of destroyed objects.

4.7 Code Sharing

CCTT SAF has separate behaviors for each force for a majority of the CISs (see Figure 1), but it also has common behaviors for both forces (e.g., Platoon Execute Traveling). This represents a trade-off between code maintainability and the need for separate behaviors. ModSAF uses the same behaviors for BLUFOR and OPFOR vehicles. This is a problem for behaviors that execute a given behavior differently. For example, Assault an Enemy Position for a Tank Platoon is executed differently for each force. The BLUFOR CIS for Assault an Enemy Position specifies CGF operator intervention and calls for moving to the Objective using covered and concealed routes. Conversely, the

OPFOR behavior requires neither operator intervention nor moving to the Objective using covered and concealed routes. Because a CIS is tailored to either BLUFOR or OPFOR behaviors, a mechanism must be introduced to provide separate behaviors for BLUFOR and OPFOR in ModSAF. One solution to this problem is to augment ModSAF's task filtering mechanism to distinguish

Two other similar areas of concern for code sharing are sharing behaviors across unit types and across echelons. In CCTT SAF two units of different types but at the same echelon level may use the same body of code. For example, in OPFOR Assault an

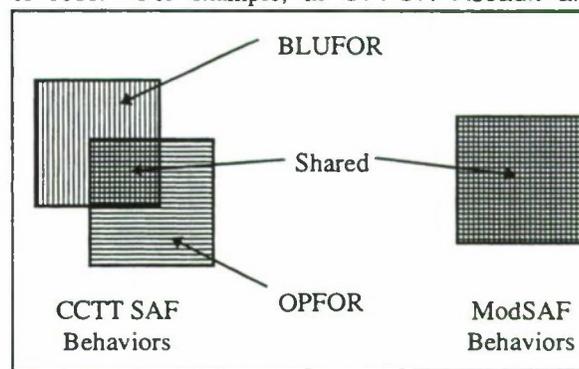


Figure 1 - BLUFOR/OPFOR Code Sharing

Enemy Position both a tank platoon and a motorized rifle platoon will execute the same body of code, but a tank company executes different code. In ModSAF, behavior code is shared between unit types, and to some extent, between echelons.

4.8 Crew-level behaviors

CCTT SAF has implemented behaviors down to the crew level. This is largely a naming convention for low-level vehicle responsibilities consisting of the Weapons Crew (e.g., target assessment), Maintenance Crew, Driver (e.g., routing), and Resupply Crew. The Crew-level behaviors in CCTT SAF order the Driver to move to a certain location, while ModSAF starts a task for a subordinate vehicle to travel to a certain location. Consequently, for implementation of behaviors in ModSAF, a design decision to either implement Crew-level behavior or use existing ModSAF functionality must be made.

4.9 Conclusions of CCTT SAF and ModSAF Differences

Although important, many of the differences between the two systems were disregarded for this project to allow implementations of CISs in ModSAF. Supporting behaviors and CGF services were similar enough to support implementation of CISs. Future interoperability enhancements to supporting behaviors and CGF services will only support more realistic behaviors.

5. Prototype Implementation

After a survey of the CATT-Task database, IST chose eight CISs to implement in ModSAF (see Table 2). These were chosen to get a sampling of behaviors that are simple, complex, BLUFOR, OPFOR, in ModSAF, not in ModSAF, for a Company, and for a Platoon.

BLUFOR	Status
Conduct Hasty Occupation of Battle Position (B0025)	Done
React to Air Attack (B0113)	
Execute Traveling Overwatch (B0017)	
Emplace Hasty Protective Minefield (B0137)	
OPFOR	Status
Conduct Fire Engagement (HVY-0324)	
Assault an Enemy Position (HVY-0022)	Done
Execute Evasive Actions (HVY-0029)	
Company Assault an Enemy Position (HVY-0113)	

Table 2 - Selected CISs As part of this research, IST manually added two CISs to ModSAF (shown in gray in Table 2). This process began by analyzing the CIS definition in the CATT-Task database and constructing flow diagrams and a list of inputs and outputs required by the CIS. The CCTT SAF code was then analyzed and compared to the CIS definition. Necessary components for the CIS were sought in ModSAF. A ModSAF implementation was designed using this gathered information. Necessary underlying code in ModSAF was used in the design. The design was then implemented and tested.

5.1 B0025 Conduct Hasty Occupation of a Battle Position

This section presents a description of Conduct Hasty Occupation of a Battle Position, outlines IST's approach to implementing Conduct Hasty Occupation of a Battle Position in ModSAF, and evaluates IST's CIS implementation.

5.1.1 CIS Description

In Conduct Hasty Occupation of a Battle Position, a U.S. Platoon moves toward and occupies a Battle Position (CATT-Task Database). A description of this CIS as given in the CATT-Task database appears in Figure 2.

Figure 2 - Conduct Hasty Occupation of a Battle

The platoon is conducting offensive or defensive operations and has received an order to conduct a hasty occupation of a battle position (BP). The platoon moves to and occupies the BP, orients itself properly on the likely direction/avenue of enemy attack and/or assigned engagement area (EA), ensures survivability of the platoon and its fighting position, and is prepared to defend the BP by the time specified in the order (ARTEP 17-237-10-MTP, pp. 5-112 to 5-114; FM 17-15, pp. 4-20, 4-4 and 4-5, 4-10 to 4-16).

BP	An advantageous location, selected on the basis of terrain and weapon systems, from which a unit defends or attacks. Platoon BPs and their direct-fire orientation are designated in the Operations Order (FM 17-15, p. 2-8).
EA	An area designated along enemy avenue(s) of approach in which the commander intends to destroy the enemy force with massed fires. It can be identified by prominent terrain features or by Target Reference Points at the corners (FM 71-2, p. 4-22).

Figure 2 - Conduct Hasty Occupation of a Battle Position

Position shows the sequence of actions that a Tank Platoon should follow in the Conduct Hasty Occupation of a Battle Position:

1. Routes to the center of the Battle Position.
2. Locates covered and concealed positions.
3. Moves to and occupies the Battle Position.

5.1.1.1 Supporting CISs for B0025

Conduct Hasty Occupy Battle Position uses other CISs for situational interrupts. The situational interrupts required for this CIS are listed below.

Situational Interrupts:

B0013	React to Indirect Fires
B0020	Take Active Air Defense While Stationary
B0022	Execute Actions on Contact

These three CISs represent actions to be taken by the Platoon executing a Hasty Occupy. React to Indirect Fires occurs when the Platoon encounters indirect fire. Take Active Air Defense While Stationary is used to respond to air threats. Execute Actions on Contact provides instructions to follow when opposing ground forces are encountered.

5.1.2 IST Approach

For this prototype, IST explored the CIS, examined the corresponding CCTT SAF code, looked at the existing ModSAF code for a similar behavior, and constructed a diagram for the IST implementation.

5.1.2.1 CIS Task Description

IST first reviewed the CIS definition. The Actions to

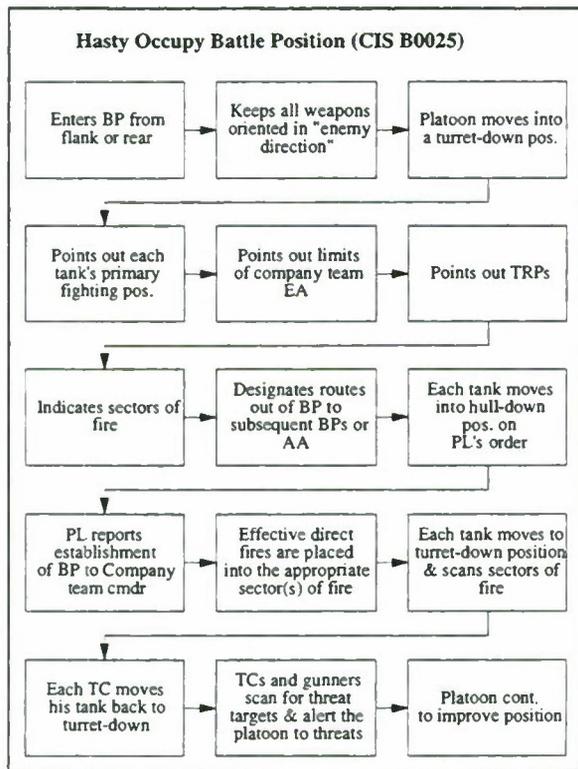


Figure 3 - CIS Description of B0025

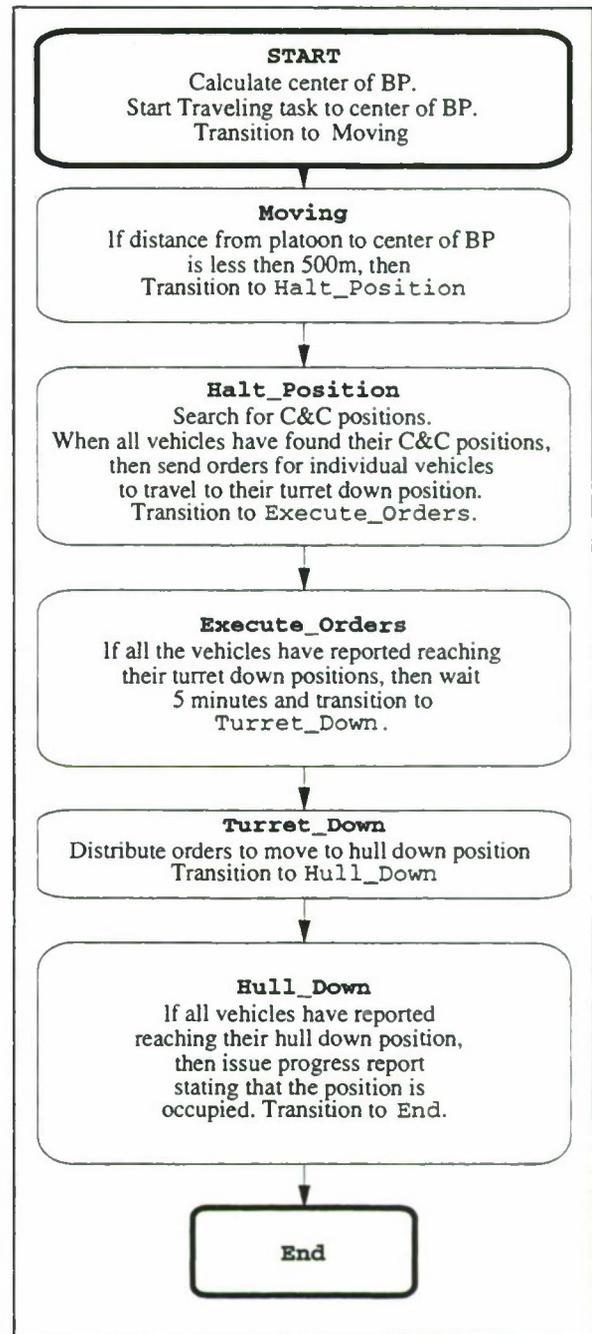


Figure 4 - CCTT SAF Implementation of B0025

be Taken section of the CIS definition outlines the procedure a unit follows when executing a CIS. IST researchers created a flow diagram from the high level actions in this section (Figure 3).

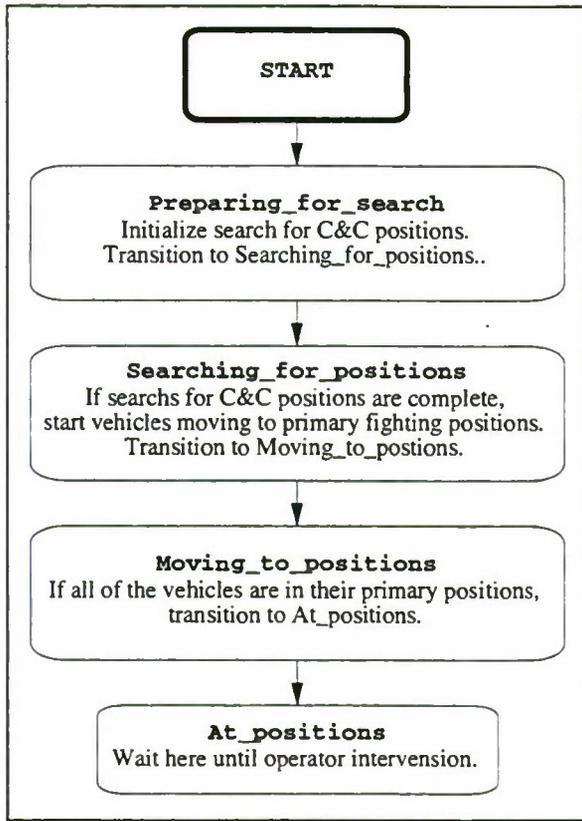


Figure 5 - ModSAF's Hasty Occupy Position

Actions from top to bottom represent the sequential order of actions that occur in the task. From the description, a platoon executing B0025 would arrive at the Battle Position. Individual vehicles of the platoon would move into a turret down position, move into primary fighting positions, move into turret-down positions, and then continue to improve their positions.

5.1.2.2 CCTT SAF Code

IST researchers created FSM state diagrams from CCTT SAF behavior code. The state diagram provides an overview of the sequence of actions in the CCTT SAF implementation of B0025, including transition timing. Figure 4 for the state diagram for CIS B0025.

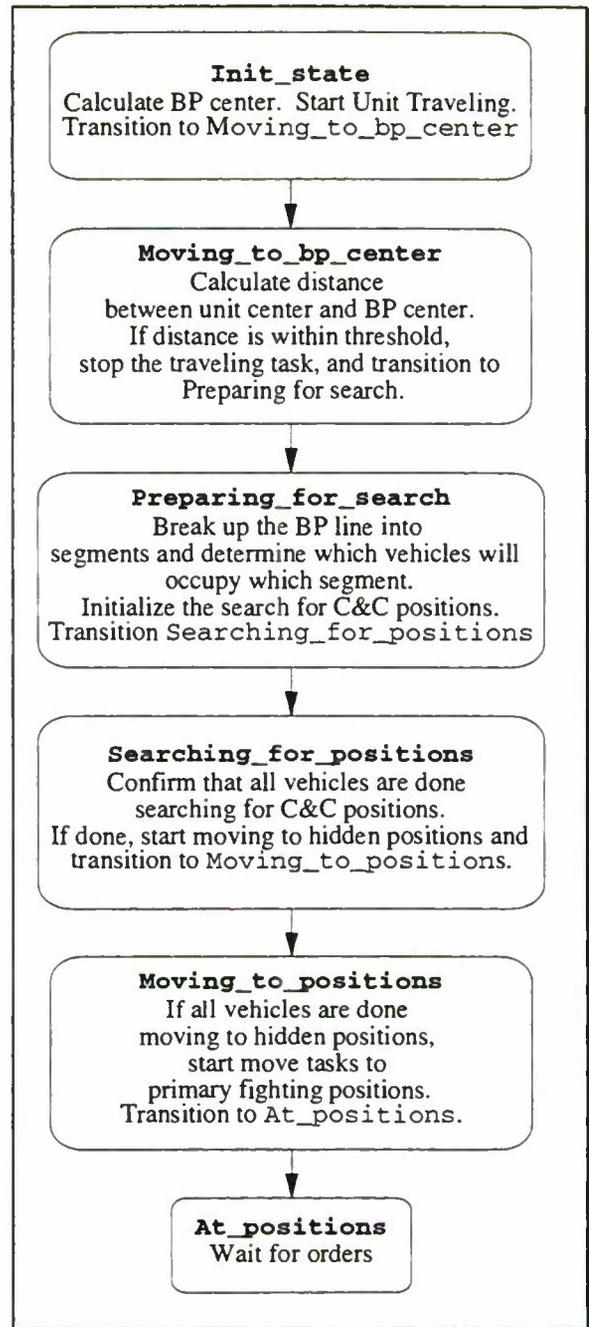


Figure 6 - Implemented Conduct Hasty Occupation of Battle Position

Figure 4 shows the actions of the CCTT SAF Hasty Occupation of a Battle Position. Each of the boxes represents a state in the CCTT SAF code. Transitions between states are represented with arrows.

5.1.2.3 Similar Existing ModSAF Behavior

A similar behavior to Conduct a Hasty Occupation of a Battle Position existing in ModSAF is Hasty Occupy Position. A state diagram for Hasty Occupy Position appears in Figure 5.

Figure 5 shows the actions of the Hasty Occupy Position. Each of the boxes represents a state in the ModSAF code. Transitions between states are represented with arrows. This ModSAF behavior was analyzed to determine the modifications necessary to implement CIS B0025 in ModSAF. Note that in Figure 4 the platoon moves as a unit to the center of the Battle Position, the vehicles move independent to their hidden positions, and then the individual vehicles move their primary fighting positions. In Figure 5 however, vehicles of the platoon move independently to their primary fighting positions.

5.1.2.4 IST Implementation

Using the information gathered in Figure 3, Figure 4, and Figure 5, IST developed an implementation plan for adding Conduct Hasty Occupation of a Battle Position to ModSAF. These three diagrams were merged into a final implementation guide, Figure 6.

5.1.3 Implementation Evaluation

IST evaluated its implementation of Conduct Hasty Occupation of a Battle Position against CCTT SAF's implementation to determine the differences between the two. IST enumerated difficulties that were encountered in implementing this CIS in ModSAF. Finally, IST compared its CIS implementation to a similar behavior in ModSAF (Hasty Occupy Position).

5.1.3.1 Differences Between CCTT SAF and ModSAF Implementations

CCTT SAF uses supporting CGF Services from CCTT SAF (e.g., CCTT SAF's target acquisition, CCTT SAF's covered and concealed location search). IST's implementation uses ModSAF's supporting CGF Services (e.g., ModSAF's target acquisition, ModSAF's covered and concealed location search).

CCTT SAF's CIS implementations are built using supporting CIS implementations (e.g., Execute Traveling, React to Indirect Fires). IST's implementation of the CIS uses ModSAF supporting behaviors (e.g., unit traveling, react to indirect fire).

5.1.3.2 ModSAF Implementation Difficulties

ModSAF's supporting tasks occasionally do not behave as expected. Periodically, vehicles traveling in the unit would get out of formation, occasionally stopping to let the rear platoon vehicle pass the third vehicle. Although some amount of flexibility is expected in formation maintenance, our SME deemed this inappropriate.

5.1.3.3 Difference From Existing ModSAF Behavior

The CIS was built using ModSAF's Hasty Occupy Position task as a base. Obviously if the original task followed the CIS definition, no work would have been necessary. It should be noted that much of ModSAF's original task was similar to the CIS definition. Main differences between the CIS prototype and the existing ModSAF behavior are the use of the CIS for only one unit type, movement to Battle Position and use of hidden positions.

ModSAF's Hasty Occupy task can be used by both homogeneous and mixed units (e.g., DI-IFV platoons). This CIS is specifically written for a BLUFOR tank platoon. All references to mixed unit tasks were removed and replaced with non-mixed unit equivalent tasks. Although the mixed tasks would have accomplished the same result, they were essentially wrappers for non-mixed tasks, and therefore unneeded for these purposes.

At the beginning of ModSAF's Hasty Occupy Position (right side Figure 7), each vehicle finds a primary fighting position, an alternate fighting position, and hidden positions. The vehicles then move individually to their primary fighting positions. In IST's CIS implementation, the center of the Battle Position is calculated. The platoon moves in column formation to this location (on left in Figure 7). After the platoon arrives, individual vehicles search for C&C positions. When all vehicles have located their positions, the vehicles individually move to their hidden positions. After waiting in their hidden positions (delay period is specified in the CIS), the vehicles move to their primary fighting positions.

When enemy vehicles are spotted,

1. the engagement area is updated,
2. new C&C positions are calculated,
3. the entire platoon moves first to new hidden positions (typically using reverse gear),
4. waits a specific amount of time and
5. moves to the new primary firing positions (typically using forward gear).

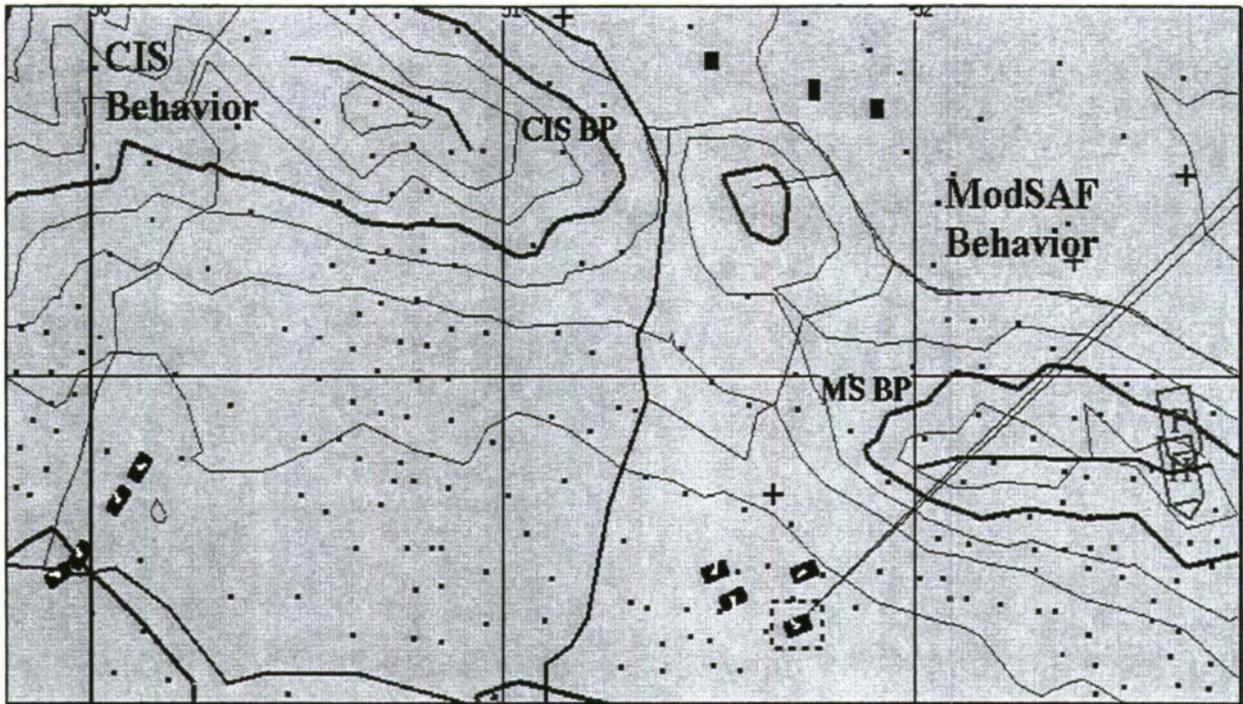


Figure 7 - CIS and ModSAF Hasty Occupy Position

Use of a hidden position is significant in that it allows better use of the vehicles front armor. The sequence of backing into a hidden position and driving forward to a new fighting position (vehicles move from P to H' to P' in Figure 8) keeps a vehicles front armor facing the engagement area. The ModSAF behavior skips steps 3 and 4 (vehicles move directly from P to P') ignoring hidden position. Consequently vehicles often expose weaker side armor while moving to a new fighting position.

6. Lessons Learned

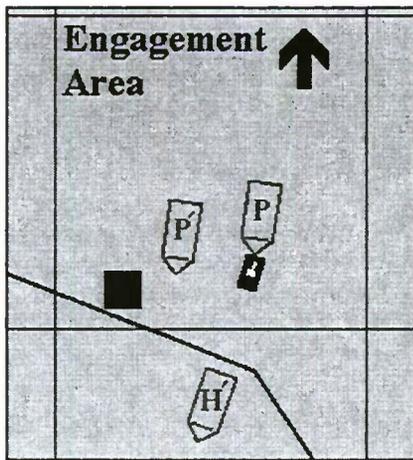


Figure 8 - Use of Hidden Position

Many of the issues uncovered initially in this project have been addressed with the implementation of these CISs. Some of these key issues include:

- Can behaviors for specific alignments be implemented in ModSAF?
- How different are ModSAF behaviors and CCTT SAF CIS based behaviors?
- Can CISs be implemented in ModSAF?
- Can the knowledge engineering and software development effort of CCTT SAF program be leveraged to help implement CISs in ModSAF?
- Can this process be automated?

Behaviors for specific alignments (OPFOR Hasty Occupation of a Battle Position, and a different BLUFOR Hasty Occupation of a Battle Position) can be incorporated into ModSAF. Behaviors for separate alignments were added to ModSAF in separate libraries. Each behavior was then restricted in use to members of the alignment for whom the task was designed (i.e., only BLUFOR vehicles could execute a BLUFOR defined CIS).

ModSAF behaviors seem to have much of the functionality required by CISs, but seem to lack many of the CIS details. This is probably due to ModSAF's use of one behavior for many types of units and

vehicles (e.g., tank platoons use the same behavior as mixed Dismounted Infantry/Infantry Fighting Vehicle platoons).

CISs can be implemented in ModSAF. IST implemented two CISs in ModSAF based on the CIS definition and the CCTT SAF code. These CISs rely on ModSAF behaviors such as unit traveling, but do perform the tasks called for in the definition.

Reusing CCTT SAF technology leverages the investment the US government has made in simulation and training. IST leveraged the CCTT SAF software development effort to help implement CISs in ModSAF.

Much of the process to implement a CIS (at least for the first prototypes) is repetitive and can be automated. For example, creating state diagrams from CIS definitions and CCTT SAF code, library duplication, and inserting CIS definitions in source code to closely tie the definition to the code could be automated. Methods to automate this process will be examined further by IST.

7. Future Work

In addition to further study of behavioral integration techniques, there are a number of important problems to be solved that are of value to the simulation community.

- *Measuring Behavioral Interoperability*- Methods are needed to measure the impact of integration issues identified above and the overall execution of specific behaviors. Behavioral interoperability needs to be clearly defined and methods developed for measuring it.
- *CGF Independent Behaviors*- CGF systems currently have unique behavior implementations. The same behavior implementations could be used in different CGF systems. One approach to this is to develop a behavior interface library. One side of the library interfaces to a CGF system and the other side interfaces to CGF independent behavior code. Two CGF systems could then execute the exact same behavior code and share in the benefits of interoperability, code re-use and validation.
- *Flexible Behavior Sequencing*- The Execution Matrix used in ModSAF and CCTT SAF utilizes a spreadsheet type form that executes a single stream of behaviors from beginning to end. Realism is compromised because it is not possible to plan out contingencies in advance and execute different behaviors without operator intervention. Research

is necessary on development of a behavioral organization that supports decision making on the fly, possibly based on METT, with execution of multiple behavioral streams.

8. Conclusion

This project has demonstrated the feasibility of adding CISs to ModSAF. This paper has illustrated the process that was used to add a CIS to ModSAF and has documented some of the general issues that impact behaviors. IST determined that many of the enumerated issues can be overcome (by IST or the CGF community). CIS prototype implementations helped reveal processes used in CIS implementation that could be automated. By examining the CATT-Task database, IST found that a core group of CISs are frequently used by other CISs. More examination of the CATT-Task database is necessary to fully enumerate this list, but this core group of CISs represents a starting point for CIS integration.

9. Acknowledgment

This research was sponsored by the U.S. Army Simulation, Training, and Instrumentation Command and by the U.S. Army TRADOC Analysis Center as part of the Integrated Eagle/BDS-D Project, contract number N61339-92-K-0002. That support is gratefully acknowledged.

10. References

- Braudaway, W., Buettner, C., Chamberlain, F., Evans, A., Smith, J., and Stanzone, T. (1995). *LibCTDB - Compact Terrain DataBase Library User Manual and Report*, ModSAF documentation.
- Campbell, C., Hull, R., Root, E., and Jackson, L. (1995). "Route Planning in CCTT", *Proceedings of the Fifth Conference on Computer Generated Forces and Behavioral Representation*, Orlando, FL, Institute for Simulation and Training, May 9-11, 1995, pp. 233-243.
- CATT-Task Database. Phase II Version 2.0. Resource Consultants Inc.
- Courtemanche, A. and Ceranowicz, A. (1995). "ModSAF Development Status", *Proceedings on the Fifth Conference on Computer Generated Forces and Behavioral Representation*, Orlando, FL, Institute for Simulation and Training, May 9-11, 1995, pp. 3-13.
- Horan, B. (1994). "A SEOD Sneak Preview (Coming Soon to a Simulator Near You)", *11th DIS Workshop of Standards for the Interoperability*

- of *Distributed Simulations*, Sept. 26-30, 1994, Orlando FL, Institute for Simulation and Training. pp. 379-388.
- Marshall, H. (1996). Personal Communication, March 8, 1996.
- Mastaglio T. and Goodwin III, E. (1994). "Integrating Users into System Development: User Exercises in CCTT", *Proceedings of the 16th Interservice/Industry Training Systems and Education Conference*, Nov. 28 - Dec. 1 1994, Orlando FL, Section 1-9.
- Pope, C., Vuong, M., Moore, R., and Cowser, S. (1995). "A Whole New CCTT World", *Military Simulation & Training*, Issue 6, pp. 6-19.
- Rajput, S. and Karr, C. (1995). *Cooperative Behavior in ModSAF*, IST Technical Report, IST-CR-95-35.
- Smith, J. and Swarts, S. (1990). *LibSched - LibSched, Programmer's Reference Manual*, ModSAF documentation.
- Smith, S. and Petty, M. (1992). "Controlling Autonomous Behavior in Real-Time Simulation", *Proceedings of the Southeastern Simulation Conference 1992*, The Society for Computer Simulation, Pensacola, FL, October 22-23, 1992, pp. 56-71.
- Vrablik, R. and Richardson, W. (1994). "Benchmarking and Optimization of ModSAF", *Proceedings of the Fourth Conference on Computer Generated Forces and Behavioral Representation*, Orlando, FL, Institute for Simulation and Training, May 4-6, 1994, pp. 25-33.
- Wright. (1994). "Source Data Acquisition for the Close Combat Tactical Training Systems and Education Conference", *Proceedings of the 16th Interservice/Industry Training Systems and Education Conference*, Nov. 28 - Dec. 1 1994, Orlando FL, Section 1-12.
- Engineering from the University of Central Florida. His research interests are in the area of simulation.
- Tracy R. Tolley** is a Research Consultant on the Integrated Eagle/BDS-D Project at the Institute for Simulation and Training. She has earned a B. S. in Mathematics from the University of Central Florida, and is currently pursuing an M. S. in Computer Science from UCF. Her research interests are in the areas of simulation, graph theory, and parallel programming.
- Lee Napravnik** is a Software Engineer in the Integrated Eagle/BDS-D Project at the Institute for Simulation and Training. Mr. Napravnik has earned a B.S. in Mathematics and B.S. in Computer Science from Missouri Western State College, and is currently a M.S. student in Computer Science at the University of Central Florida. His research and publications are in the areas of simulation and WWW software development.
- Daniel E. Mullally** is a Research Associate for the Institute of Simulation and Training with over twenty years of military experience followed by over 18 years of training-related experience. Mr. Mullally holds a M.A. in Human Resources Management from Pepperdine University and a B.S. in Business Administration from Western Carolina University.
- Robert W. Franceschini** is a Computer Scientist at the Institute for Simulation and Training. He currently leads the Integrated Eagle/BDS-D project at IST. Mr. Franceschini has earned a Bachelor of Science in Computer Science from the University of Central Florida; he is currently pursuing a Master of Science in Computer Science from UCF. His research interests are in the areas of simulation, graph theory, and computational geometry.

11. Authors' Biographies

Matthew K. Kraus is an Associate in Simulation at the Institute for Simulation and Training. He has a Bachelor of Science degree in Computer Science from Western Michigan University. He is currently pursuing a Master of Science degree in Simulation Systems at UCF. His research interests are in the areas of distributed computing, artificial intelligence, and computer graphics.

Derrick J. Franceschini is a Research Consultant at the Institute for Simulation and Training. He earned a Bachelor of Science degree in Computer

Semantic Arbitration of Behavior for the Interoperability of SAF Simulations

Frederic McKenzie
Rick_McKenzie@cpqm.saic.com
Science Applications International Corporation
3045 Technology Parkway
Orlando, FL 32826

Christopher Dean
deanc@orl.saic.com
Science Applications International Corporation

Avelino Gonzalez
ajg@engr.ucf.edu
Dept. of Electrical and Computer Engineering
University of Central Florida

1. Abstract

The Advanced Distributed Simulation Research Team (ADS RT) at SAIC-Orlando has been conducting experiments with the interoperability of simulations. One of these experiments focuses on a generic approach for sharing behaviors between Modular Semi-Automated Forces (ModSAF) and Close Combat Tactical Trainer Semi-Automated Forces (CCTT-SAF). One goal of military simulation training is to provide large scale or joint exercises to train personnel at higher echelons. To help meet this goal a research experiment was designed to investigate how lower echelon combatants may consist of computer generated forces with units composed of entities represented by different simulations and different SAF operators/facilitators. This research explores a method of reducing operator work load by allowing units of one simulation to be task organized with high echelon units of a different type of simulation. In this way, the simulation that owns the higher echelon unit would be able to impose behaviors on unfamiliar subordinate units without the aid of an operator. The result is a unit composed of different simulations that behave as one unit. This is accomplished through the correlation of the behaviors of entities in different simulations so that they can cooperate with one another while performing unit tasks. Specific behaviors from the simulation with the upper echelon unit (source behaviors) can be translated to a form in terms of general behaviors which can then be correlated to the behaviors of any desired subordinate unit owned by a different simulation (destination behaviors) without prior knowledge of the pairing. The approach is to develop an ontology of general behaviors and behavior parameters, a database of behaviors written in terms of these general behaviors,

and heuristic metrics which are used to compare source behaviors with destination behaviors.

The results of this research has shown that heuristic metrics, in conjunction with a corresponding behavior and parameter ontology, are sufficient for the correlation of heterogeneous simulation behavior. These metrics successfully correlated known pairings provided by experts. In addition, the metrics also provided reasonable correlations for behaviors that have no corresponding destination behavior. For an environment composed of a variety of SAF participants, these metrics show great promise to serve as a foundation for more complex methods of arbitration. A description of the generic arbitration algorithm and the results of the experiments are contained in this paper.

2. Introduction

The initial focus of Distributed Interactive Simulation (DIS) application development has been on training of large, joint, or combined forces which is lacking in traditional training. (DIS Steering Committee, 1994). Since no single simulation can meet all the training needs required for large or joint exercises, multiple simulations must be used that can interoperate with one another seamlessly in a common environment. The DIS protocol was developed to promote interoperability in a heterogeneous simulation environment. Experience has shown that the DIS standards do not address all of the issues associated with interoperability. Although DIS provides standards and guidance for interface definition, communication, environment representation, management, security, field instrumentation, and performance measurement, it does not specify entity representation standards, behavior standards, synchronization standards, or spatial coherence

(correlation of terrain, resolution correlation and environment correlation such as ambient illumination, buildings, weather, etc.) standards and database standards. This research specifically addresses the behavior standards problem and the behavior interoperability of SAF simulations.

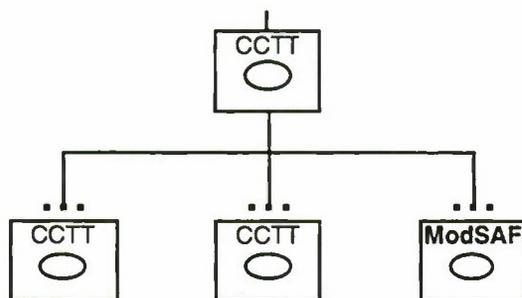


Figure 1: Task Organized Heterogeneous Simulation Units

3. Behavior Interoperability

Because of the military's desire to conduct large-scale theater of war training exercises and joint force operations, there is a growing interest in the use of SAF in the generation of simulated forces. Coordination between different services employing different SAF systems requires that the SAF systems be capable of coordinated actions. To help alleviate SAF operator workload in such an environment, CGF units must be able to be composed of entities that are owned and simulated by different simulations (Figure 1). The units must perform their actions properly under the specified task organization, i.e. each unit must coordinate with every other unit even if simulated by different simulations. This can be a problem since the behavior of the simulations may be of a different fidelity or functionality. Also, different simulations may not even possess corresponding behaviors. Behavior interoperability addresses these interactions in an attempt to achieve the same performance from the different simulations, i.e. behavior correlation. Using this approach, entire missions need to be arbitrated regardless of the method used (reactive, intelligent agents, CBR, etc.).

To address the problem of behavior interoperability, a common framework is necessary to provide a basis for correlating SAF behaviors (Smith, 1995). In object-oriented terms, simulation entities, events, etc. can be converted from their specific form to a general form and then to the specific form required by the destination simulation. The extra step of converting to the general common model provides flexibility in that it allows interoperability between different combinations of simulations without having to know the exact combination beforehand. For the correlation of behavior, not only is a common behavior

framework necessary but some degree of correlation of the behavior is required that can allow a simulation to execute the behavior specified by a another simulation. This requires that all the necessary attributes of a behavior must be imitated by both simulations. However, due to the differences in simulation behaviors, the behavior correlated for the destination simulation may not be exactly the same as that of the source simulation, i.e. they do not share a common framework. Thus, the best match or correlation must be arbitrated. Only major changes to the destination simulation's architecture (to support a common framework) would allow the total correlation specified by the above definition. For this research, arbitration will be defined as the discrimination of behaviors based on an evaluation of semantically correlated tactical procedures between two heterogeneous simulations. A discriminated behavior will be the best fit tactical maneuvers for a subordinate unit based on the requirements of its commanding unit. Semantic arbitration for behavior needs to not only arbitrate the best "match" between simulation behaviors, but also correlate the parameters associated with the behaviors. If the parameters of the commanding unit's behavior cannot be correlated with the target simulation behavior then the behavior cannot be executed.

As part of the subject research, a methodology was developed that promotes interoperability of behavior among simulations using a common behavior framework, along with heuristic metrics to correlate behavior. A set of closeness heuristic metrics has been defined for both behaviors and their parameters. These metrics will use the general behavior and parameter ontologies to determine the destination behavior with the best "semantic closeness" to the given source behavior.

To satisfy the problem of interoperable SAF simulations, this research involves the development of a general framework for behavior and behavior parameters that facilitates the correlation between tactical procedures. The structure of this framework is domain independent which enables the system to be used with other applications outside Department of Defense training. Additionally, the system may be used to perform off-line arbitration between known simulations and parameter correlation during run-time or the system can perform arbitration at run-time to allow any combination of simulations to interoperate.

4. Semantic Correlation

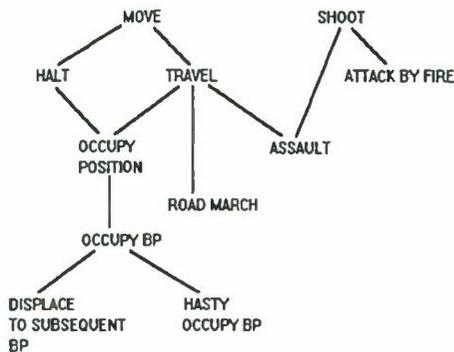


Figure 2: Partial Hierarchy for Tank Platoon Behaviors

4.1 Behavior Correlation Metrics

Behaviors are usually represented in an aggregate fashion. Higher level behaviors are represented in terms of lower level behaviors until the primitive level is reached. Behaviors may be represented in terms of more general behaviors or the aggregate of lower echelon behaviors. In the case of aggregate lower echelon behaviors, different behaviors may be assigned to different units. This is not a problem since the higher echelon behavior can still be considered to exhibit these behaviors even though not all lower echelon units exhibit all the behavior. Because there is an infinite number of ways the same behavior can be represented a simple comparison is not sufficient. When trying to compare and correlate behaviors several metrics can be used to determine how similar they are:

- A source behavior can be found at a lower or higher level of decomposition of a behavior than in the destination behavior. This is defined as the WHERE-IS metric.
- A source behavior can be decomposed into its sub-behaviors which can then be correlated. This is defined as the HAS-A metric.
- A source behavior can be related to a more general or more specific behavior present in the destination behavior. This is defined as the IS-A metric. Note that an ontology such as that shown in Figure 2 is necessary for this determination.
- A source behavior can be related to a similar behavior of the destination. This is defined as the SIBLING-OF metric.

Any combination of these metrics can be used at the various levels of decomposition to determine the semantic closeness of two behaviors. In this context, semantic closeness is defined as the percentage that the destination behavior will perform the desired behavior. There is no guarantee that the chosen behavior will execute the same behavior as the source, only that it will be the best match possible among the available destination behaviors. Many times, behaviors may be essentially the same but are organized differently. There are five major cases that illustrate the various ways differently structured behaviors can be correlated. The five cases use contrived examples of behavior from the military domain for the sole purpose of illustrating the possible metrics. The behaviors of interest in each case are represented in *italics*.

The first case illustrates a source behavior that is found one deeper level of decomposition on the destination side. If the behavior is not found, then its subcomponents can be used as a means of correlation. An example of the first case is:

CASE 1: Lower Level WHERE-IS

<u>Behavior A:</u>	<u>Behavior B:</u>
<i>TRAVEL</i>	CAUTIOUS-MOVE
<i>MOVE</i>	<i>TRAVEL</i>
OCCUPY-POSITION	OCCUPY-POSITION

Case 2 illustrates a similar situation but in reverse, the behavior is found two levels of decomposition higher:

CASE 2: Upper Level WHERE-IS

<u>Behavior A:</u>	<u>Behavior B:</u>
ASSAULT	ATTACK-BY-FIRE
<i>TRAVEL</i>	TARGETER
OCCUPY-BP	<i>TRAVEL</i>
<i>TRAVEL</i>	<i>OCCUPY-</i>
TARGETER	<i>POSITION</i>
<i>OCCUPY-</i>	
<i>POSITION</i>	
CONSOLIDATE	CONSOLIDATE

Case 3 illustrates the situation where the behavior is decomposed into its sub-behaviors and correlated:

CASE 3: HAS-A

<u>Behavior A:</u>	<u>Behavior B:</u>
ASSAULT	TRAVEL
TRAVEL	OCCUPY_BP
OCCUPY_BP	TARGETER
TARGETER	TRAVEL
TRAVEL	OCCUPY-
OCCUPY-	POSITION
POSITION	
CONSOLIDATE	CONSOLIDATE

Case 4 illustrates both the general-to-specific and specific-to-general IS-A correlation. When correlating from behavior A to behavior B the more specific HASTY_OCCUPY_BP can be used in place of OCCUPY_BP. When correlating from behavior B to behavior A, the more general OCCUPY_BP can be used in place of HASTY_OCCUPY_BP. Case 4 is as follows:

CASE 4: IS-A

<u>Behavior A:</u>	<u>Behavior B:</u>
ASSAULT	ASSAULT
TRAVEL	TRAVEL
OCCUPY_BP	MOVE
TRAVEL	TARGETER
MOVE	HASTY_OCCUPY_BP
SHOOT	OCCUPY-POSITION
CONSOLIDATE	CONSOLIDATE

Case 5 illustrates the SIBLING-OF correlation. Here BOUNDING_OVERWATCH is correlated with TRAVELING_OVERWATCH since they are inherited from the same parents, and hence similar:

CASE 5: SIBLING-OF

<u>Behavior A:</u>	<u>Behavior B:</u>
ASSAULT	ASSAULT
BOUNDING-	TRAVELING-
OVERWATCH	OVERWATCH
TRAVEL	TRAVEL
OCCUPY-	OCCUPY-
POSITION	POSITION
OCCUPY-	OCCUPY-
POSITION	POSITION
CONSOLIDATE	CONSOLIDATE

Extra behaviors may also be present on either the source behavior or destination behavior. Extra behaviors on the

destination behavior do not affect the closeness as it has been defined. Extra behaviors only mean that the destination behavior does more than needed which is acceptable. Only if the extra behaviors drastically cause the behavior to conflict with the source behavior will there be a problem. There may also be some ambiguity if more than one destination behavior share the same subset of behaviors that match the source behavior. As far as the semantic closeness is concerned the behaviors are equal. A modification to the algorithm could be made that would choose the behavior with the least amount of extra behavior but that is no guarantee that behaviors will not be ambiguous. Extra behaviors on the source behavior do decrease the closeness since the destination behavior may be missing some important functionality.

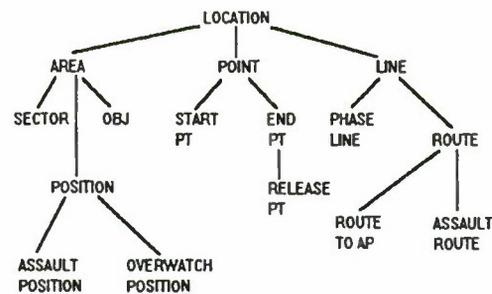


Figure 3: Partial Hierarchy for Behavior Parameters

4.3 Parameter Correlation Metrics

In addition to performing metrics when correlating behaviors, metrics must also be calculated for correlating the parameters associated with that behavior. Parameters either are necessary for the corresponding behavior to perform its function or modify how the behavior is executed. Common parameters for military behaviors include speed, formation, platform, route, etc. The metrics define how close the parameters between the two behaviors match. Parameter correlation is only performed for the top level source and destination behavior. The parameters of sub-behaviors are not really significant since as long as the initial parameters correlate, the behavior can be executed. In addition, many times the sub-behavior parameters will be derived internally and have no explicit relationship to the top level parameters.

There are three metrics that apply to parameter correlation, the IS-A, PARENT-OF and HAS-A metrics. The IS-A and PARENT-OF metrics both determine the closeness along an inference path between a source parameter and destination parameter using information shown in Figure 3. The IS-A metric determines if a destination parameter is a child of one of the source parameters. The metric determines the inferential distance between the two.

Similarly, the PARENT-OF metric determines if a destination parameter is a parent of one of the source parameters. Unmatched (Additional) parents in a PARENT-OF metric also do not affect the closeness for the parameter. This just means that the parameter is more complex than the source parameter being correlated which is satisfactory. These two metrics can be combined to generate a correlation path from a specific source parameter to a more general parameter and then back to a more specific destination parameter. For example, an ASSAULT_POSITION can be correlated to an OBJECTIVE by following the inference path from ASSAULT_POSITION to POSITION to AREA to OBJECTIVE, where OBJECTIVE is a specific type of AREA. The HAS-A metric determines the closeness along a decomposition path between a source parameter and destination parameter. For example, suppose a ROUTE can be decomposed into a START_POINT and END_POINT. Then, a source ROUTE parameter can be correlated with START_POINT and END_POINT parameters of the destination behavior. The IS-A and PARENT-OF metrics can be combined with the HAS-A metric so that the sub-parameters of parameter may also be matched with destination parameters.

4.4 Incremental Decomposition And Abstraction

The correlation algorithm uses incremental decomposition and abstraction of behaviors to determine the closeness. Each source behavior is recursed into and is compared (via recursion again) to the levels of the destination behavior. Each behavior is decomposed into its sub-behaviors which are also correlated down to the primitive level. The correlation algorithm uses the following high level steps when correlating a source behavior:

- 1) Check for the presence of the sourcebehavior at the given level of decomposition in the destination behavior.
- 2) If the behavior is not present, apply the WHERE-IS, IS-A, HAS-A, and SIBLING-OF metrics, using the maximum closeness result.
- 3) Recurse into the source behavior, performing these steps on each sub-behavior. Combine the results of the sub-behavior correlations and multiply the result by the closeness value determined in one of the two previous steps.
- 4) Repeat steps 1-3 on the next behavior at this same level of decomposition.

The parameter correlation algorithm follows the same basic steps, with the parameter metrics being applied instead. It is important to note that behaviors can

increase the closeness if they match, but behaviors that match in name are not necessarily equal. The closeness must be determined down to the primitive level to determine an accurate correlation (hence the presence of step 3 above). The correlation algorithm uses the semantic closeness metrics defined earlier to determine the behavior closeness value. This value is calculated using closeness factors (decreases in closeness) for each metric along with a few others. These factors may need to be adjusted for a specific destination system to guarantee proper correlation.

As each source behavior is correlated, the metric that produces the best closeness value is combined with the aggregate closeness value of its sub-behaviors. The value is then combined with the other behaviors at the same level of decomposition and filtered up to the upper levels of decomposition. At the top-level, the correlation of the behaviors is combined with the parameter correlation to obtain a final correlation for the behavior in the range between 0 and 1. Each sub-behavior (except reactive behaviors) are equally important in the closeness determination. Reactive behaviors count for less since they do not define the behavior, only their presence helps determine the closeness. The algorithm makes sure that it does not recurse into reactive behaviors when looking non-reactive source behaviors since this would drastically throw off the correlation. Also, a destination sub-behavior can be correlated against a source behavior more than once. In some cases this makes sense and is useful if a destination behavior encapsulates more of the source behavior. However, in some cases this is not true. The uncertainty is captured by the decrease in closeness factor for the correlation but no decrease in correlation is currently implemented for multiple destination matches.

The parameter correlation mechanism is a simpler form of the behavior correlation algorithm. As mentioned previously, this is primarily because it is focused on a conversion path not just similarity. The WHERE-IS metric is not used since sub-parameters on the destination side are never recursed into. Source parameters are broken up into their constituents if necessary and these are matched against the top-level destination parameters only. Missing parameters contribute a portion of the closeness if they are default. Unmatched required parameters on the destination side will set the entire behavior closeness to zero, because even if the behaviors are similar, if the parameters cannot be correlated then the behavior cannot be executed. Unmatched required source parameters only decrease the closeness determination by setting their closeness contribution to zero. Both source and destination parameters that are default and cannot be correlated are not set to zero only the closeness is reduced by a

specified amount. Default parameters are defined as those which have preset values within their appropriate simulations and are not required to be set for the behavior to be executed.

This research focused on the correlation of CCTT tank platoon behaviors with that of ModSAF tank platoon behaviors so they could interoperate under one task organization. Only those behaviors that could be assigned to tank platoons via their respective GUIs were considered for source and destination correlation. Each CCTT behavior assigned to a ModSAF platoon would be correlated

with the best matching ModSAF behavior and its parameters converted and the behavior assigned.

5.1 Proof Of Principle

As a proof of principle, twelve CCTT behaviors will be correlated with one of twenty ModSAF behaviors. Seven of these behaviors will have expected pairings provided by subject matter experts. The remaining five will have no corresponding ModSAF behavior. The unknown correlation results are subject to interpretation since no agreed correlation already exists. Table 1 presents the correlations that will be tested via the experiments.

CCTT-MODSAF CORRELATIONS

CCTT BEHAVIOR	MODSAF BEHAVIOR
Assault an Enemy Position	Assault
Attack by Fire	Attack by Fire
Bounding Overwatch	Overwatch Movement
Tactical Road March	Tactical Road March
Travel	Travel
Hasty Occupy Position	Hasty Occupy Position
Traveling Overwatch	Traveling Overwatch
Occupy Bp	<i>unknown</i>
Passage of Lines	<i>unknown</i>
Platoon Defensive Mission	<i>unknown</i>
Platoon Fire and Movement	<i>unknown</i>
Consolidate and Reorganize	<i>unknown</i>

Table 1

5 Experimental Results

Testing ASSAULT and ATTACK BY FIRE will test the algorithms ability to discriminate between similar offensive actions. Testing BOUNDING OVERWATCH will test the algorithms ability to discriminate between several ModSAF forms of movement, namely TRAVEL, TACTICAL ROAD MARCH, OVERWATCH MOVEMENT, and TRAVELING OVERWATCH. A similar reason

applies to TACTICAL ROAD MARCH. The ModSAF TACTICAL ROAD MARCH is not as robust and thus may not be determined to be the best correlation. Testing TRAVEL will set the lower bound for the test since this behavior exhibits a strong correlation to the ModSAF TRAVEL behavior.

5.2 An Experiment

One experiment involves correlating the CCTT Assault An Enemy Position behavior. A typical tank platoon assault behavior is concerned with issuing movement and firing commands to its vehicles. These commands instruct the vehicles to perform an on-line attack and occupy the position attacked. More specifically, the tank platoon closes with and destroys the enemy by overrunning and seizing the occupied enemy position. The tanks move rapidly in line formation under the cover from direct and indirect fire to the far side of the objective. Figure 4 shows the CCTT Assault An Enemy Position behavior. CCTT is more robust than ModSAF in this case because it provides for an initial travel to the assault position, allows for the breach of obstacles along the way, and a consolidation and reorganization of forces after the assault has been completed.

CCTT ASSAULT ENEMY POSITION:

```

TRAVEL
  vehicle_MOVE
BOUNDING_OVERWATCH
  TRAVEL
    vehicle_MOVE
  vehicle_OCCUPY_POSITION
  vehicle_MOVE
  vehicle_SEARCH
  vehicle_HIDE
    vehicle_HALT
    vehicle_MOVE
    vehicle_SEARCH
SEEK_COVER_AND_CONCEALMENT
  vehicle_SEARCH
  vehicle_OCCUPY_POSITION
...
GENERATE_REQUEST_FOR_IFIRE
CONSOLIDATE_AND_REORGANIZE
  SEEK_COVER_AND_CONCEALMENT
  vehicle_SEARCH
  vehicle_OCCUPY_POSITION
...
GENERATE_SITREP
  
```

Figure 4: CCTT Assault An Enemy Position

For the CCTT Assault An Enemy Position behavior, the following semantic closeness values were calculated for the ModSAF behaviors:

ASSAULT	0.522923
ASSEMBLE	0.264287
ATTACH	0.425562
ATTACK BY FIRE	0.39817
BREACH	0.431557
CHANGE FORMATION	0.265716
CONCEALMENT	0.401982
DELAY	0.419041
DETACH	0.425562
FOLLOW VEHICLE	0.0
HALT	0.264287
HASTY OCCUPY POSITION	0.377462
OVERWATCH MOVEMENT	0.43608
PLOW BREACH	0.431557
PURSUE	0.0
ROAD MARCH	0.422094
SUPPLY	0.400714
TRAVEL	0.422094
TRAVELING OVERWATCH	0.488249
WITHDRAW	0.409559

The highest correlation is with the ModSAF assault behavior with a semantic closeness of 52%. The actual closeness value is not so important as is the relative values between the different ModSAF behaviors. This pairing is the expected correlation.

The semantic closeness values of zero represent cases where required ModSAF parameters could not be correlated. Figure 5 shows the ModSAF Assault behavior. The common primitives of vehicle_MOVE and vehicle_SEARCH (common to OCCUPY_POSITION) and the TRAVEL behavior are the primary reasons for the correct correlation. For similar reasons, the second and third choices (TRAVELING_OVERWATCH and OVERWATCH_MOVEMENT, respectively) exhibited high semantic closeness values. The presence of these primitives in several OCCUPY_POSITION behaviors offset some of the missing behaviors even though the positions being occupied are very different. The different positions are captured by the parameter correlation but their effect on the overall closeness is much smaller.

MODSAF ASSAULT:

```

EXECUTE_ASSAULT
ASSAULT
  TRAVEL
    vehicle_MOVE
    vehicle_SEARCH
    vehicle_ENEMY
  FOLLOW_UNIT
    vehicle_MOVE
    vehicle_SEARCH
    vehicle_ENEMY
  TARGETER
    vehicle_SHOOT
    vehicle_ASSESS
    vehicle_SEARCH
  OCCUPY_POSITION
    vehicle_ALTERNATE
    vehicle_MOVE
    vehicle_TERRAIN
    vehicle_SEARCH
  
```

Figure 5: ModSAF Assault Behavior

The CCTT parameters were correlated with the ModSAF parameters in the following fashion with their corresponding closeness values:

- CCTT UNIT_ID to ModSAF UNIT_ID (SC = 1.0)
- CCTT PLATFORM to ModSAF PLATFORM (SC = 1.0)
- CCTT ROUTE_TO_AP to ModSAF ROUTE (SC = 0.9)
- CCTT ASSAULT_ROUTE to ModSAF ROUTE (SC = 0.9)
- CCTT ENEMY_POSITION to ModSAF POSITION TO AREA (SC = 0.729)

CCTT TRIGGER_LINE to LINE to ModSAF
 ROUTE (SC = 0.81)
 CCTT ASSAULT_POSITION to POSITION
 to AREA to ModSAF OBJECTIVE
 (SC = 0.729)
 CCTT DEPARTURE_TIME to NO MATCH
 (SC = 0.0)
 CCTT OBSTACLE defaulted (SC = 0.9)
 CCTT BREACH_ROUTE to ModSAF
 ROUTE (SC = 0.9)
 CCTT PRE-BREACH_ROUTE to ModSAF
 ROUTE (SC = 0.9)
 CCTT POST-BREACH_ROUTE to
 ModSAF ROUTE (SC = 0.9)
 CCTT ALPHA_SECTION ignored
 (SC = 0.75)
 CCTT BRAVO_SECTION ignored
 (SC = 0.75)
 ModSAF LEFT_TACTICAL_BOUNDARY
 defaulted (SC = 0.75)
 ModSAF RIGHT_TACTICAL_BOUNDARY
 defaulted (SC = 0.75)
 ModSAF SPEED defaulted (SC = 0.75)
 ModSAF DISMOUNTED_SPEED defaulted
 (SC = 0.75)
 ModSAF
 STOPPING_ASSAULT_CRITERIA
 defaulted (SC = 0.75)
 ModSAF SECURE_OBJECTIVE_FLAG
 defaulted (SC = 0.75)
 ModSAF FORMATION defaulted
 (SC = 0.75)
 ModSAF SPACING defaulted (SC = 0.75)
 ModSAF X_DI_OFFSET defaulted
 (SC = 0.75)
 ModSAF Y_DI_OFFSET defaulted
 (SC = 0.75)
 ModSAF ASSAULT_REASON defaulted
 (SC = 0.75)
 ModSAF DI_FORMATION defaulted
 (SC = 0.75)

The results agree with the predictions with one exception that illustrates one inherent problem with the parameter correlation. Destination parameters that are equally related to more than one source parameter

cause an ambiguity as to which parameter correlation is the correct one. In this experiment there are five equally related source routes and only one destination route. We know that the ASSAULT_ROUTE is the best correlation but it is unclear as to how the algorithm can determine this automatically. Correlating in the other direction, a single source behavior can be matched against more than one destination behavior. In some cases this may be satisfactory but in other cases it may cause unexpected results and thus the destination parameters should have been allowed to default. Some a priori knowledge code may need to be used to modify the parameter correlation for known problems before assigning the behavior. As an example, code can be used that will check to see if all the routes are the same and if they are, default all the routes except the assault route. Also, the best correlations should take precedence over lesser correlations such as the TRIGGER_LINE in this case. The CCTT TRIGGER_LINE should be ignored since there are better ROUTE correlations. This is a trivial task that can be done when the actual parameter conversions are done. The ordering of the parameters may also be used to specify a priority as a conflict resolution scheme. However this may not always be correct when the simulations being correlated is determined at run time.

5.3 Experiment Conclusions

Based upon the results of the experiments, it has been shown that the use of heuristic metrics in conjunction with a corresponding behavior and parameter ontology is sufficient for correlating CCTT and ModSAF behaviors. Table 2 summarizes the results of the experiments. Out of seven expected correlations, six were correlated correctly with the one exception due to a deficiency in ModSAF. The remaining five unknown correlations were deemed acceptable by subject matter experts under the given constraints. Most of the correlations resulted in closeness values around 50% thus demonstrating the dramatic differences that can be present in externally similar systems.

SUMMARY OF EXPERIMENTAL RESULTS

#	CCTT SOURCE	MODSAF RESULT	MODSAF EXPECTED	SEMANTIC CLOSENESS	ACCEPT-ABLE
1	ASSAULT ENEMY POSITION	ASSAULT	ASSAULT	0.522923	YES
2	ATTACK BY FIRE	ATTACK BY FIRE	ATTACK BY FIRE	0.607225	YES
3	BOUNDING OVERWATCH	OVERWATCH MOVEMENT	OVERWATCH MOVEMENT	0.554897	YES
4	TRAVELING OVERWATCH	TRAVELING OVERWATCH	TRAVELING OVERWATCH	0.744768	YES
5	TACTICAL ROAD MRCH	BREACH	TACTICAL ROAD MRCH	0.51583	NO
6	TRAVEL	TRAVEL	TRAVEL	0.899357	YES
7	CONSOLIDAT REORGANIZE	DELAY	<NONE>	0.489362	YES
8	OCCUPY BP	ASSAULT	<NONE>	0.589559	YES
9	PASSAGE OF LINES	TRAVELING OVERWATCH	<NONE>	0.39317	YES
10	PLATOON DEFENSIVE MISSION	ASSAULT	<NONE>	0.540253	YES
11	PLATOON FIRE AND MOVEMENT	ASSAULT	<NONE>	0.528677	YES
12	HASTY OCCUPY POSITION	HASTY OCCUPY POSITION	HASTY OCCUPY POSITION	0.594519	YES

6. Conclusion

This research has shown that SAF behaviors can be correlated with behaviors from different simulations so they can interoperate with one another to support simulation training. Specific source behaviors are translated to a form in terms of general behaviors which are then correlated to any desired specific Table 2

destination simulation behavior without prior knowledge of the pairing. As the experiments show, the correlation may not be 100% since the simulations may have different semantics. The experiments do show that the use of heuristic metrics in conjunction with a corresponding behavior and parameter ontology is sufficient for correlating heterogeneous simulation behavior.

This research has shown that using a database of CCTT behaviors and ModSAF behaviors written in a general form, a common ontology of behavior parameters, and a set of heuristic metrics, that CCTT and ModSAF tank platoons can interoperate (to a degree) under one task organization. Of the seven known pairings experiments, six showed the expected

results. Even though the correct ModSAF behaviors were selected, however, many of the closeness values were quite low. This is further proof of how simulations that appear similar externally can actually be very different in their internal semantics. As mentioned previously, the one failed experiment was not due to an error in the correlation algorithm but due to the drastic difference in robustness of the supposedly the same behavior. The five unknown pairings produced acceptable results (as determined by experts) when considering that there was no corresponding ModSAF behaviors for these CCTT behaviors. The ModSAF and CCTT units are still interoperating but not to the degree desired. Often 100% interoperability of like simulations (same class such as virtual or constructive) requires complete reengineering of one of the simulations to the extent that it is no longer beneficial to use two different simulations at all.

This research has shown that a less sophisticated form of correlation with a simple behavior representation can indeed correlate behavior correctly and satisfactorily in most cases. This has the potential to reduce the SAF operator workload in

large-scale exercises. It also demonstrates the promise of using heuristic metrics and knowledge frameworks to solve semantic interoperability problems. As the state of the art in CGF increases, these semantic interoperability issues will become the dominant factor in the pursuit of large-scale and joint exercises. This research is but the first step towards the heterogeneous simulations of the future.

7. Future Work

The focus of this research has been on the arbitration algorithm and its supporting components. The actual run-time interfaces and parameter conversion routines have yet to be developed. There were also several issues addressed in the arbitration algorithm. Specifically, how to handle source parameters that correlate to more than one destination parameter equally, and destination parameters that correlate to more than one source behavior. Both can cause unexpected behavior when the behavior is executed with these parameter conversions. Also, more research is required to study when to allow parameters default instead of being correlated. Of course, if 100% correlation is desired than an extension of this work is needed that allows simulations to be data driven and share behavior primitives.

8. Acknowledgements

The authors would like to thank Christina Bouwens for her support on this paper.

9. References

- Dean, Christopher, *The Semantic Correlation of Behavior for the Interoperability of Heterogeneous Simulations*, Masters Thesis, University of Central Florida, May 1996.
- DIS Steering Committee, *The DIS Vision: A Map to the Future of Distributed Simulation Version 1*, Institute for Simulation and Training, May 1994.
- Smith, Roger D., "The Conflict Between Heterogenous Simulations and Interoperability," *Proceedings of the 17th Interservice/Industry Training Systems and Education Conference (CD-ROM)*, The American Defense and Preparedness Association, Albuquerque, NM, November 13-16, 1995.

10. Authors' Biographies

Dr. Frederic (Rick) McKenzie has been a member of the Advanced Distributed Simulation Research Team (ADS RT) since April 1995 serving as P.I. for two interoperability IRAD projects. He holds a Senior Scientist position at SAIC Orlando and is currently

technical lead on an ARPA sponsored advanced interoperability project. For two years prior to joining the ADS RT, he had been a member of the knowledge engineering team for the SAF component of the Close Combat Tactical Trainer (CCTT) project. Dr. McKenzie has had two years teaching experience in software languages and data structures. He obtained a Master of Science in Computer Engineering in 1990 and a Ph.D. in Engineering in 1994 from the University of Central Florida. Both his Masters and Ph.D. work have been in AI research, focusing on knowledge representation and model-based diagnostic reasoning.

Christopher Dean has been a member of the Advanced Distributed Simulation Research Team (ADS RT) since January 1995. He has been involved with several R&D efforts and is currently supporting an ARPA sponsored advanced interoperability project involving the verification and validation of SAF behaviors. Christopher has seven years of prior development experience creating commercial educational software using various software engineering techniques including OOA/OOD, process improvement, and documentation. Christopher has a Bachelors of Science in Computer Engineering from the University of Florida in 1994 a Masters in Computer Engineering from the University of Central Florida in 1996. His Masters work has focused on knowledge based systems and object oriented techniques and their application to simulation training.

Dr. Avelino J. Gonzalez received his Bachelor's and Master's degrees in Electrical Engineering from the University of Miami, in 1973 and 1974 respectively. He obtained his Ph.D. from the University of Pittsburgh in 1979 also in Electrical Engineering. He spent nearly 12 years with Westinghouse Electric Corp. in Pittsburgh, PA and Orlando, FL working in computer applications to electrical power engineering. One of his most significant efforts was the development of the Westinghouse generator diagnostic expert system GenAID for which he received the Westinghouse Award of Excellence in Engineering.

In 1986, Dr. Gonzalez joined the Computer Engineering Department at the University of Central Florida (UCF) in Orlando, where he has focused his research and teaching in Artificial Intelligence and knowledge-based systems.

Dr. Gonzalez is a registered Professional Engineer in the State of Florida, and was the founding president of the Florida Artificial Intelligence Research Society. He presently serves as the Treasurer of that organization.

Generating Computer Generated Forces

Robert Balzer
Information Sciences Institute
4676 Admiralty Way, Marina Del Rey, CA 90292
balzer@isi.edu

1. Abstract

There is nothing special about our Computer Generated Forces, except that they are automatically generated from high level descriptions in a formal specification language.

We have designed a high level graphical language for specifying military doctrine concerning the makeup and behavior of military forces and a synthetic force generator which translates these specifications into simulation modules that fit into the ModSAF architecture and simulate the behavior of those military forces. Central to this specification language is the definition of formations describing the coordinated movement and behavior of a disaggregated military group.

This language is intended to eventually cover the full range of military doctrines found in the four services and was inspired by the Combat Instructions Sets developed by the Army for the CCTT simulator. Those validated doctrines are stated in natural language. Our specification language attempts to cover the same space and require little semantic restructuring to formally specify those doctrines.

The initial version of this language has focused on the definition of military formations and the movement and maneuvers of the individual units in that formation. It has been used to formally specify the naval doctrine for how convoys are refueled and resupplied at sea and to automatically generate the ModSAF code from this specification.

2. Introduction

The objective of our effort is to design a formal specification language for describing synthetic force behavior which is semantically close to the information contained in the Army's Combat Instruction Sets (CISs) [McEnany & Marshall 1994] and a generator for that language which automatically converts those formal specifications into operational ModSAF code.

2.1 Maneuver Specification Language

We have developed an initial prototype of such a language, called Maneuver, which allows formations to be graphically defined. These formations identify the participants in the formation, specify their locations relative to one another, and indicate the movements or other actions they should perform in the formation together with any synchronization or other constraints on those actions.

A specification consists of a set of such formations and rules for switching between them including the reassignment of roles between those formations. These rules specify for each participant what role it plays in the new formation and what maneuvers or other actions it must undertake to reach its assigned position in that formation.

Bounding Overwatch can thus be specified through two formations which respectively switch the (stationary) "overwatch" and (advancing) "bounding" roles between two squads.

2.2 Maneuver Generator

We have also built a generator which translates Maneuver specifications into operational ModSAF code in two stages. In the first stage, these specifications are translated into a high level general purpose specification language called Relational Abstraction [Goldman & Naraswamy 1992]. This high level general purpose specification language was designed to facilitate the definition of the formal semantics of domain specific languages (such as Maneuver) while delaying decisions about how that semantics would be implemented.

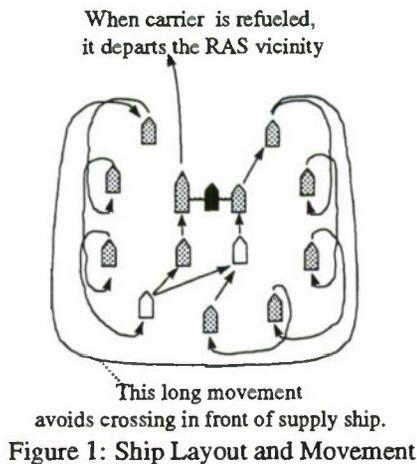
Those implementation decisions are addressed in the second stage of translation which occurs in alternative back-ends that produce operational code in a particular programming language. We have previously developed Relational Abstraction back-ends for Lisp, C++, and Ada, and have utilized the first two in this effort.

3. Replenishment At Sea Example

3.1 Informal Specification

To illustrate the scope of the existing language, we (informally) summarize the naval doctrine for refueling a naval convoy at sea:

A formation is described (see Figure 1) in which the convoy ships are arrayed in a circle around the supply ship (shown in black) and rotate from position to position to eventually take up a refueling position along side the supply ship (on either its left or right side). The left and right hand semi-circles (as seen from the heading of the convoy) rotate independently of each other. This rotation is triggered by the pulling away of the refueled ship on that side of the supply ship. The length of time it takes for a ship to refuel is determined by the amount of fuel that it needs and the number of supply hoses that it has for the refueling (only a single size hose is used). In addition, the ship must hook-up those hoses before refueling can begin, and break those connections before pulling away.



Eventually, all of the ships in one of the semi-circles will be all be refueled. At that point, the ships from the other side will begin to rotate into both refueling positions (rather than just the single one on their side of the supply ship). This is accomplished as a change in formation. The two formations share most of their roles. The formation change is specified by naming the target formation and specifying the target roles for those ships which are *changing* roles. Each formation has its own rules for when synthetic entities switch roles, how those role switches are synchronized with others, and how the synthetic entities move from one position to another. (In the

Replenishment at Sea task ships move from one position in a semicircle to another by pulling forward and turning to the outside of the formation, proceeding to the rear until they pass their intended position, and then turning back to the inside and pulling up into their new assigned position. During these maneuvers they are forbidden from crossing in front of the supply ship).

Variant formations are defined for describing how the carrier of the convoy takes up a special position outside the rotation once it has been refueled. Additional, formations are used to describe the optional use of helicopters to ferry containerized packets of material from the supply ship to convoy ships abeam of the supply ship (but not those currently refueling). These ships move closer to the supply ship to facilitate this helicopter resupply

3.2 Formal Specification

All the information contained in the previous section – except for the details of the path taken in moving from one position to another -- is formally specified in the Maneuver language through a set of annotated diagrams and a small number of non-graphical textual declarations. The Replenishment At Sea task without helicopters is defined by 5 formations (one of which is terminal) and 10 textual declarations. Specifying the behavior of the helicopters requires an additional 2 formations (one of which is terminal). In addition, to specify the behavior when an (unpredictable) emergency breakdown occurs, 4 more formations are required.

One diagram (see Figure 2) specifies the layout of the initial formation of the ships in the convoy around the supply ship and defines all of the positions in that formation that can be occupied by ships during the Replenishment At Sea task. Besides the circle of positions at a 1 to 2 mile radius around the supply ship these positions include the refueling positions on either side of the ship (LF and RF), the waiting positions (LW and RW) 500 yards aft of these refueling positions (in which ships are prepositioned as the refueling of the ship on that side is nearing completion so that when that ship pulls away the next ship to be refueled can quickly pull into position), and the carrier position (C) several miles ahead of the convoy where the carrier goes after being refueled (to get itself out of the way of all the movement and maneuvering occurring during this task).

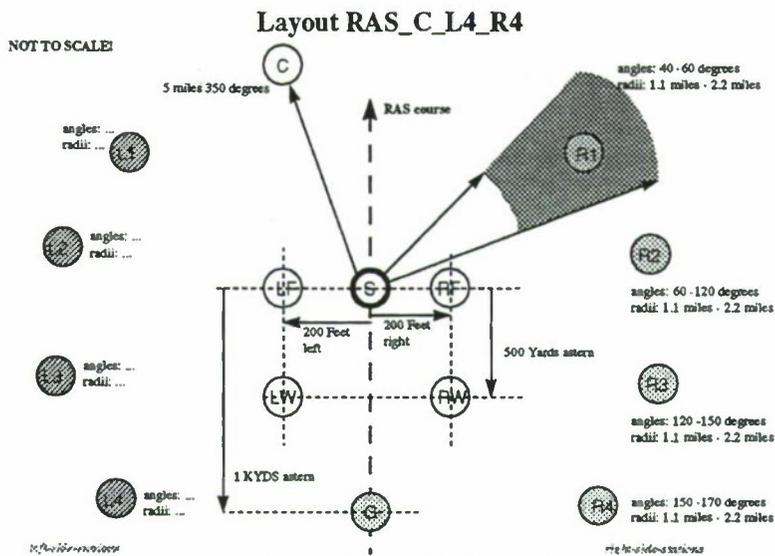


Figure 2: Initial Formation Layout

Each of the other diagrams specifies the behavior for a single formation including the rules, if any, for switching to another formation (specified by another diagram). Figure 3 specifies the behavior of the initial formation. It is initialized with the carrier and one ship at the left and right waiting stations (LW & RW), with the rest of the convoy ships in the circle of positions arrayed around the supply ship (L1-L4, R1-R4, and G). The refueling positions (LF and RF) and the carrier refueled position (C) are unassigned.

Each of the arrows in a behavior diagram specifies a role switch (and movement) that can occur within the specification. These role switches are governed by the predicates, if any, that appear on the arrow, but can only occur when the role at the arrow's tail has an assigned ship. In Figure 3 the LW and RW roles are occupied in the initial state, and the predicate on their outward-bound arrows ("not asg?" meaning "not assigned") is true. Therefore, these two role switches can occur. However, there is

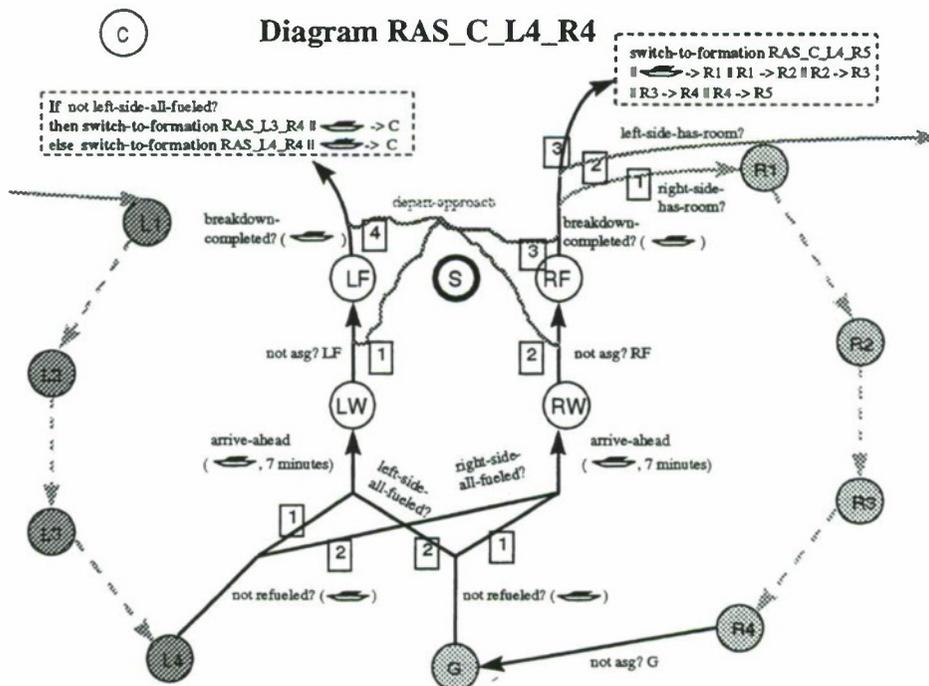


Figure 3: Initial Formation Behavior

another constraint (“depart-approach”) on these transitions (indicated by the wavy line that connects the transitions into and out of the refueling positions LF and RF).

This synchronizing constraint prevents arrivals and departures alongside the supply ships from overlapping each other, so the navigator of the supply ship can focus on a single arriving or departing ship. It is defined textually in Figure 5. The numbered boxes on the constraint connectors indicate preference among the possibilities. Here those preferences allow the carrier in LW to be reassigned to LF and move into the left refueling position and block the reassignment of the ship in RW to RF until one minute after the arrival of the carrier at the left refueling position.

Once a ship has arrived at the left or right fueling stations it hooks up its fueling hose(s), fuels, and then disconnects (breakdown) those hose(s). This triggered sequence and the definitions of these three actions are textually specified in Figure 5.

The completion of the breakdown for one of the refueling ships triggers (via the “breakdown complete?” predicate) the reassignment of that ship to one of the positions emanating from the arrow out of that fueling position. As with the arrival arrows into the fueling positions, these departure arrows are synchronized by the “depart-approach” constraint which will delay this reassignment as necessary to prevent its overlap with other arrivals or departures.

For the right fueling station (RF) the choice of which of the three positions to transit to from the refueling position is governed by the priorities specified in the numbered boxes attached to the arrows leading to the three possible targets. Each of these is tried in turn until a true predicate is found. Since the “right side has room” (i.e. as defined in Figure 5, not all the right hand positions are occupied --- which will be true until all the right hand side ships have been refueled and rotated out of the refueling station), the highest priority choice will be available and the refueled ship departing RF will be reassigned to R1.

The dashed arrows emanating from roles R1, R2, and R3 indicate that these transitions are synchronized with the reassignment to the role at the start of the arrow. Thus, when a ship assigned to RF is reassigned to R1, R1’s ship is reassigned to R2, R2’s ship is reassigned to R3, and R3’s ship is reassigned to R4. The rotation on the left hand side

(L1 to L4) is similarly synchronized (by the dashed arrows) with the reassignment of a ship to L1.

The reassignment of a ship to the lifeguard position (G) is not synchronized with the reassignments from R1 to R4. Rather it is governed by the predicate attached to its transition arrow (“not asg? G”). Thus, whenever G is unassigned and R4 is assigned, the ship assigned to R4 will be reassigned to G. (Reassignment may occur even before a ship arrives at a destination, so a ship in transit from R3 to R4 may be reassigned to G if G is vacated before that ship arrives at R4.)

As long as the ship at the lifeguard position (G) is not refueled, that ship will be reassigned to the left or right waiting position (LW or RW) when the ship at the corresponding refueling station is nearing completion of its refueling (formally it will be reassigned in anticipation of that completion so that the waiting ship will be in position at the waiting station 7 minutes before the refueling ship’s breakdown is complete). The right hand waiting station is the preferred choice but ships can also be reassigned to the left hand waiting station once all the ships on the left side have been fueled. The reassignments into and out of L4 are defined similarly to those into and out of the lifeguard position.

All of the transitions in Figure 3 have been explained except for the transition from LF and the second and third choices from RF. Let’s start with the transition from LF. It specifies a switch of formations to formation RAS_L3_R4 (see Figure 4) or formation RAS_L4_R4 (not shown) depending on whether the left side still has room. In the unusual case when it doesn’t (because the refueling of the carrier took so long that the right hand side was completely refueled and all the left hand side ships were refueled on the right hand side of the supply ship), the convoy switches to the terminal RAS_L4_R4 formation. Normally, when the rest of the convoy has not yet been completely refueled, it switches to the RAS_L3_R4 formation.

This formation is very similar to the starting formation, except that it has one fewer left hand side position (L1 through L3 instead of L1 through L4) to account for the fact that the carrier has been reassigned to position C and is not participating in the left hand side rotation. The reassignment of the carrier (the ship icon denotes the ship assigned to the role at the start of the reassignment arrow) to position C is explicitly stated. By default all the

Diagram RAS_L3_R4

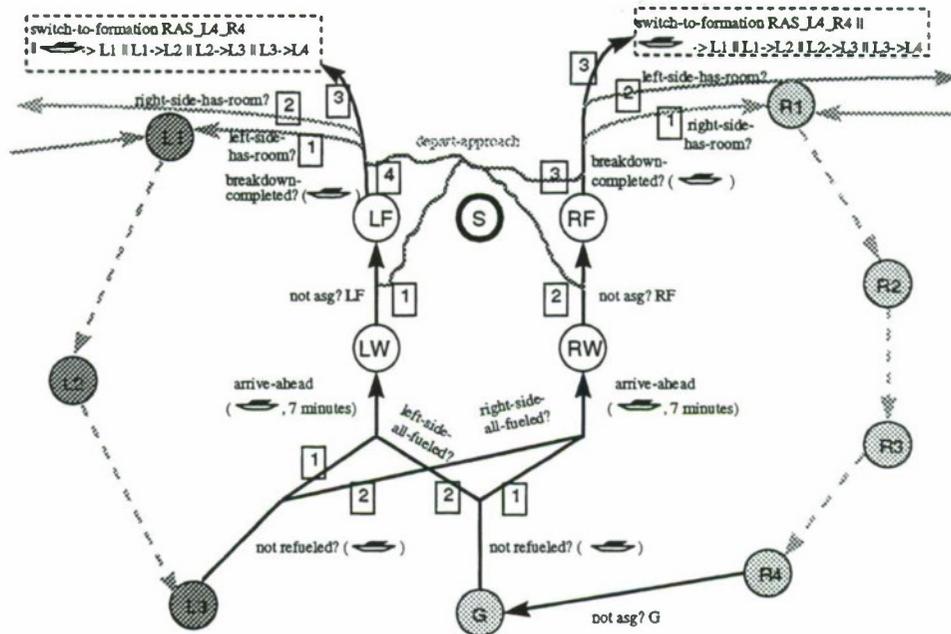


Figure 4: Left Side Short Formation

remaining (unmentioned) roles are carried forward to the new formation. However, the relative *positions* of those roles in the new formation may be (and in this case, some are) different, so the ships may have to maneuver to satisfy the specification.

Returning to the second and third choices from RF,

the second choice will be selected when room remains on the left side but not on the right (that is, when all the ships on the right hand side have been refueled and reassigned away from the refueling position, but some on the left remain to be refueled). In these cases, RF is reassigned to L1 and the left hand side rotates in synchronization with this

further activities not specified in the diagram:

```

on arrival at RF fueling station, hookup, fuel, and then breakdown
upon @?RF do {hookup[@RF];fuel[@RF];breakdown[@RF]}
              abort emergency? do emergency-breakdown[@RF]
upon @?LF do {hookup[@LF];fuel[@LF];breakdown[@LF]}
              abort emergency? do emergency-breakdown[@LF]

```

prevent undesirable overlaps:

```

these events must not overlap; reassignments will be delayed to assure this
1st minute of departures approaches and 1st minute thereafter
overlap depart-approach (S1,S2,S3,S4:ship) =
{ from S1:LF-> to S1:LF-> + 1 minute,   from S2:RF-> to S2:RF-> + 1 minute,
  from S3:LW-> to S3@LF + 1 minute,     from S4:RW-> to S4@RF + 1 minute }

```

action definitions:

```

simulate hookup of a ship at a station by waiting 20 minutes.
definition hookup[ship] = wait 20 minutes
definition fuel[ship] = wait (0.95 * (fuel-capacity(ship) - fuel-remaining(ship))
                             / fuel-rate(ship));
                             assert fueled?(ship) simulate filling to 95% of capacity
definition breakdown[ship] = wait 15 minutes; assert breakdown-completed?(ship)
definition emergency-breakdown[ship] = wait 5 minutes;
                                     assert breakdown-completed?(ship)

```

predicate definitions:

```

definition left-side-all-fueled? =(every left-side-stations, s, is fueled?(asg(s)))
definition right-side-all-fueled? =(every right-side-stations, s, is fueled?(asg(s)))
True if every ship assigned to a left (or right) side station is fueled.
definition right-side-has-room? = not (every right-side-stations, s, is asg?(s))

```

Figure 5: Textual Declarations

reassignment. Finally, if all ships on both sides have been refueled *except* the carrier, the third choice from RF is selected and a switch is made to a formation which creates a new right hand position for the refueled ship and rotates the ships on that side into their final positions.

The operation of the RAS_L3_R4 formation in Figure 4 is very similar to the RAS_C_L4_R4 formation that we have just described, except that it leads to a slightly different terminal formation as the third choice for the LF and RF transitions.

Although we haven't mentioned it previously, an emergency breakdown (caused by an enemy attack, an accident, or an impending collision of the supply ship) can occur at any time. If this occurs, refueling stops immediately, the refueling hoses are explosively disconnected and the refueling ships immediately depart from the resupply ship. Many of the remaining formations, which we have not shown, result from the unpredictable timing of this exceptional event.

4. Architectural Issues

4.1 Synthetic Force Commanders

This task operates as a commander for the synthetic force (convoy) which controls and coordinates the actions of the entity level units (ships) within that synthetic force. It monitors the activity occurring within the synthetic force, determines what actions should be performed by each entity level unit, and assigns the appropriate task to that unit to accomplish the required action. Thus, when breakdown is completed at one of the fueling stations, the commander task determines which option should be selected by the departing ship and assigns it a movement task that corresponds to its new role in the (current or new) formation. The commander also assigns movement tasks to any other ships whose rotation (movement) is synchronized with the ship departing the fueling station.

Thus, the commander is the only task that understands and effects the synchronization and coordination defined in the specification. The commanded units merely get a succession of primitive actions they are capable of performing individually. The coordination of the synthetic force arises from the commander's determination of the

timing and duration of those individual tasks for each of the entity level units it controls.

4.2 Abstract Target Architecture

As an interface between the Synthetic Force Commander and the entity level units that compose that synthetic force there must be a set of behaviors that those entity level synthetic forces can perform. These behaviors provide the means by which the commander can control and coordinate the actions of the entity level units that comprise the synthetic force.

This formulation of the interface between a synthetic force commander and the entity level units that comprise that synthetic force allows us to restate the generation task more precisely: given a specification that describes the desired behavior of an aggregated synthetic force, the generator must translate that specification into sets of commands to be issued by a synthetic force commander to the entity level units of that synthetic force which ensure the coordination of their behavior in performing the specified task.

Thus, the generator must understand the semantics of the behaviors that the entity level units can perform and how sequences of those behaviors can be used to accomplish the specified task. Rephrasing this in conventional compiler terminology, these behaviors are the operations of the abstract virtual machine for which programs must be generated.

We believe that the definition of that abstract virtual machine will be one of the major results of this effort. By defining a full compliment of platform specific actions (e.g. move, turn, follow, aim, shoot, reload, refuel) an operational infrastructure will be developed that supports many different synthetic force tasks (i.e. Combat Instruction Sets).

Although the code for the platform specific actions in this abstract virtual machine could theoretically be automatically generated from a specification, we have chosen not to do so, but rather to use the existing manually developed implementations produced by the military services or their contractors.

This abstract virtual machine thus forms the boundary between the automatically generated commander code and the platform specific actions that the entity level units can perform.

4.2.1 Incomplete Abstract Virtual Machine

Not surprisingly, we found mismatches between our conceptualization of a suitable abstract virtual machine and the set of available manually coded platform specific actions.

Some of these mismatches arose because our efforts were concurrent with the creation of the platform specific actions and while they eventually would be produced, they weren't ready when we needed them. Others arose from the task oriented nature of the development which limited each contractor to only produce those platform specific actions required for the SAF tasks it was assigned to implement. This resulted both in overly specific tasks and the absence of required actions. One example of the former is the inability to turn a ship to a specified heading, and an example of the latter is the absence of a following action for helicopters.

We resolved these mismatches by manually constructing code which realized the idealized abstract virtual machine operations in terms of the available implementations. Thus, to achieve a smooth "outside" turn to shift positions in the semicircle, our command task issued a sequence of course/speed adjustment commands. After each command, the command task would monitor the ships progress to decide when to issue the next course adjustment. Similarly, the absence of a following task for helicopters was resolved by regularly updating the destination location that the helicopter is moving toward (but never reaching).

A second example of an overly specific action is ship collision avoidance. It used an implementation developed for tanks. Collision avoidance is defined as maintaining a separation of x units between the boundaries of the entities. Those boundaries are calculated as one half of the entity's longest dimension. For relatively square platforms, such as tanks, this approximation works quite well. However, for long narrow platforms, like ships, it prevents them from pulling up along side of one another --- a maneuver required for refueling. Without rewriting this platform specific algorithm, our only resolution was to specify the refueling positions (LF,RF) at distances that would require unrealistically long hoses to transfer the fuel.

4.2.2 Command and Simulation

Within ModSAF, there are two possible target implementations for a "command" task like RAS.

One is to actually simulate communications between the commander and the ships - e.g., by transmitting radio messages, a concept supported by the simulator. This, requires that the ships be executing suitable tasks to monitor for radio messages, and that a message protocol exist whereby the content of the radio messages is properly interpreted by the ships to alter their behavior.

If one's goal is not to simulate the communication itself, but only the synchronized movements of the ships, a simpler mechanism is available (which we used). That mechanism allows the commander task to directly add tasks (or modify the parameters of existing tasks) in the task lists of the ships being commanded.

5. Status

There are two implementations of our generator. The first was a feasibility prototype that illustrated the possibility of automatically generating synthetic forces from a high level domain specific formal language. This generator produced Lisp code that operated outside of ModSAF in a separate process and communicated with it via network sockets.

Our second version produces C code that operates within ModSAF and eliminates the need for any run-time Lisp code. This version has been completely implemented and is being debugged. We expect to deliver shortly an operational C-based Replenishment At Sea module to NRAD's Navy SAF project.

Currently the graphic specifications must be hand translated into a set of assertions (in our Relational Abstraction language) which describe the topology of the diagrams and the labels attached to the nodes and lines. A graduate student is building a graphic editor and translator which will allow these diagrams to be interactively created and modified and will automatically construct the assertions required by the Maneuver generator.

6. Future Work

Starting this summer, we will be testing the breadth of this language and its program generator by developing formal specifications and synthesized ModSAF Computer Generated Forces for 10 CCTT Combat Instruction Sets for Scout and Mechanized Infantry Platoons that were selected by STRICOM as

being representative of a spectrum of complexity and difficulty in implementation.

7. Acknowledgment

This work was jointly funded by DARPA's ISO and ITO offices under contract DABT63-91-K-0006.

8. References

- McEnany, B.R. and Marshall, H. (1994) "CCTT SAF Functional Analysis" Fourth Conference on Computer Generated Forces and Behavioral Representation. May 1994
- Goldman, Neil and Narayanaswamy N. (1992) "Software Evolution through Iterative Prototyping" Proceedings of the 14th International Conference on Software Engineering, Melbourne, Australia, May 1992

9. Author's Biography

Robert Balzer received his B.S., M.S., and Ph.D. degrees in Electrical Engineering from the Carnegie Institute of Technology in 1964, 1965, and 1966. After several years at the Rand Corporation, he left to help form the University of Southern California's Information Sciences Institute (USC-ISI). He is currently Professor of Computer Science at USC and Director of ISI's Software Sciences Division. The Division combines Artificial Intelligence, Database, and Software Engineering techniques to automate the software development process. Current research includes program generators, architecture description and refinement, domain specific systems, transformation-based development, computing environments, constraint-based systems, and executable specification languages.

A New Mechanism for Cooperative Behavior in ModSAF

Sumeet Rajput and Clark R. Karr
Institute for Simulation and Training
3280 Progress Dr., Orlando, FL 32826
srajput@ist.ucf.edu

1. Abstract

Like vehicles in a real battlefield, CGF vehicles must cooperate with each other to achieve battlefield objectives. Traditional CGF systems, such as Modular Semi-Automated Forces (ModSAF), contain a Centralized Control Architecture (CCA) to control the behavior of simulated entities. In this approach, an entity (sometimes invisible) controls the behavior of other entities. CCAs are easy to implement but do not mirror cooperation of vehicles in the real world. A more realistic way to control the cooperative behavior of entities is through a Decentralized Control Architecture (DCA). In this approach entities cooperate with each other directly; there is no supervisory control. DCA's have several advantages and reflect real world cooperation. This paper describes a DCA developed within the ModSAF CGF system. The core of the DCA is a Finite State Machine (FSM) Engine. Cooperative behaviors are expressed as formal FSMs to obtain an unambiguous control process.

2. Cooperative Behavior

2.1 Real World Cooperation

In a real battlefield, soldiers and vehicles (actually soldiers inside the vehicles) cooperate in most situations. They may cooperate:

- by coordinating movement and fire,
- by understanding the unit's plan and their role within it,
- by reacting to unexpected events in acceptable ways,
- through information passing, and
- by following commander's directives.

A unit in the battlefield has a hierarchy of command which reflects the information flow from the top to the bottom levels.

In a real battlefield, entities cooperate in a *decentralized* fashion as opposed to using a *centralized* approach. Decentralized means that

entities cooperate with each other directly without being directly controlled by a supervisor. This does not mean they are unsupervised but rather the supervisor (commander) controls his subordinates through orders and signals and not through direct immediate control of the subordinate's behaviors.

Soldiers and vehicles cooperate either *explicitly* or *implicitly*. Explicit cooperation involves transmission of signals. Platoons transmit signals using: messenger, wire, visual, sound, and radio (US Army [1990]). Implicit cooperation does not involve any transmission of signals. Entities observe other entities and change their behavior accordingly; for example, entities do formation-keeping by observing the behavior of other entities.

2.2 Statement of the problem

The goal of the research described in this paper is to implement a CA architecture that:

- mirrors real life cooperation between vehicles,
- uses explicit and implicit cooperation between vehicles,
- allows new cooperative behaviors to be created easily and with little coding, and can be verified and validated easily.

3. Cooperative Behavior Control Architecture

The cooperative behavior control architecture controls the behavior of subordinate entities. There are two ways to control subordinate entities: Centralized control and Decentralized control.

3.1 Centralized Control Architectures (CCA)

In a Centralized Control Architecture (CCA), a centralized controller makes behavioral decisions for subordinate entities and conveys these decisions to the subordinates. CCAs resemble the real world because, like the real world, the unit is controlled from a centralized location. However, there are important distinctions. The first distinction is in the

granularity of control relative to that of the real world. CCAs exercise unrealistically fine control.

For example, CCAs may do formation-keeping for a platoon by monitoring each entity and making sure that entities maintain appropriate distances between them. In the real world, formation-keeping is done by entities; proper entity-to-entity distances are determined and maintained by entities themselves. The second distinction is in reasoning and decision making. In CCAs, the centralized controller reasons and makes decisions on the entities' behalf whereas in the real world entities reason and make decisions themselves. For example, CCAs plan routes for the entities whereas real world entities plan their own routes.

The entity exercising centralized control may be either a simulated entity (e.g., a tank) or an invisible "ghost" entity (or process). Furthermore, the centralized controller may control subordinate entities either *explicitly* or *implicitly*. Explicit control requires the transmission of messages (orders) from the centralized controller to the subordinates. These orders, unlike real world orders, contain specific information which otherwise would have been computed by the subordinates themselves. For example, an order to move may *contain* the route information. In the real world, a subordinate will only be told to move to a destination and it will compute the route itself. Implicit control is more direct. In this case, the centralized controller executes code on or on behalf of subordinate entities. Code execution directly affects a subordinate's behavior.

The centralized controller in ModSAF is an invisible process representing the unit which "knows" the identity of the vehicle responsible for the unit, e.g., a Platoon Commander. When the Platoon Commander is disabled, ModSAF restarts the cooperative behavior on the platoon. The responsible entity is updated, i.e., another entity becomes the Platoon Commander. The centralized ModSAF controller controls the subordinates implicitly by executing code on their behalf.

CCAs are suitable for implementing simple cooperative behaviors but have several disadvantages. First, implementing a CCA results in loss of realism. For example, with a "ghost" centralized controller, the unit's collective behavior can be unaffected by the loss of the simulated commander. On the other hand, if a simulated centralized controller is destroyed, the collective behavior of the unit is disrupted. Of

course, both problems can be addressed by introducing provisions in the software for transfer of command. But the complexity required to centrally resolve all the conflicts between centrally controlling a real world decentralized control process forces compromises and simplifications. To make up for these losses would entail increasing the complexity of the software. Second, generating the behaviors of all entities from a single source results in inefficient use of resources; more time is spent in the controller causing it to be overworked. Finally, modeling larger units, such as companies or battalions, becomes increasingly complex because the centralized controller has to control more vehicles.

3.2 Decentralized Control Architectures (DCA)

In a Decentralized Control Architecture (DCA), subordinate entities follow the unit's plan and commander's orders but make their own behavior decisions. Unlike a CCA, there is no unseen controller that makes decisions on their behalf; this approach mirrors cooperation in the real world. A DCA commander functions like a real world commander by giving and receiving orders from other entities. For example, a DCA commander may order an entity to move to a destination. Like the real world, the commander may only supply the entity with the location of the destination and not a precise route. In this case, the entity computes its route to reach the destination.

DCA's have several advantages. First, unit or group cooperative behavior *emerges* as a result of direct cooperation between entities potentially resulting in realistic cooperative behavior in complex situations. Second, because behavior generation is distributed across entities, which can be distributed across computers, limited hardware resources can be used efficiently. Finally, DCAs give rise to *modular* implementations; e.g., a Platoon Commander's cooperative behavior can be housed in a module separate from modules containing behaviors of other commanders. Each module's behavior can be verified and validated independently. On the other hand, CCA implementations combine the behaviors of different levels in a unit into one module making verification and validation more difficult.

When discussing DCAs two questions need to be answered. First, how do entities cooperate with each other? Second, how do entities know what task to do and when to do it?

Entities can cooperate in a number of ways:

- *Message Passing*: (Noreils [1993]), (Noreils [1992a]), (Noreils [1992b]), (Noreils [1992c]), (Parker [1994]), (Shin and Epstein [1990]), (Lefebvre and Saridis [1992]), (Smith and Davis [1981]), (Fisher and Woodridge [1994]), (Decker [1987]), (Ohko et. al. [1993]), and (Parker [1994]).
- *Shared Memory*: (Laengle and Lueth [1994a]), (Laengle and Lueth [1994b]), (Corkill [1991]), (Occello and Demazeau [1994]), and (Dai et. al. [1993]).
- *Combination of Message Passing and Shared Memory*: (Lun and Macleod [1992]), (Wang [1994]), and (Harmon et. al. [1986]).
- *Implicit Cooperation*: (Payton and Dolan [1991]).

Entities can be allocated tasks by:

- *Negotiation*: (Noreils [1993]), (Noreils [1992a]), (Noreils [1992b]), (Noreils [1992c]), (Lun and Macleod [1992]), (Smith and Davis [1981]), (Fisher and Woodridge [1994]), (Decker [1987]), and (Ohko et. al. [1993]).
- *Self-contribution*: (Corkill [1991]) and (Parker [1994]).

A survey of DCAs and the above two questions are discussed in Rajput and Karr [1995].

4. A Decentralized Control Architecture using Finite State Machines

4.1 Approach

To model cooperative behavior, IST chose to implement a DCA within the ModSAF CGF system. Traditional CGF systems have used CCAs for controlling cooperative behavior. The work described in this report is the first time a DCA has been implemented within a CGF system for controlling cooperative behavior.

The DCA chosen is based on Finite State Machines (FSMs). FSMs were used as building blocks for the architecture; FSMs are a well understood formal process control mechanism (Sudkamp [1988]).

Entities cooperate explicitly by exchanging simulated radio messages (Signal PDUs) and implicitly by observing other entities. Observation is implemented by an *Observation Module*, a software module that observes the battlefield situation and sends observation messages to the entity.

The implementation is data driven and allows new behaviors to be defined quickly and easily through data files. In current CGF systems, considerable coding effort is required to create new cooperative behaviors. This increases development and prototyping time for new cooperative behaviors.

4.2 Formal FSMs

A formal FSM is defined as:

1. A set of *states*: An FSM is in one of its states. The state of an FSM is also the state of the process being controlled by the FSM.
2. *Events*: *Only* events cause an FSM to change states.
3. *State Transition Procedures (STPs)*: These are procedures (i.e., code) which are called to do work. STPs are used *only* during state transitions.

FSMs have been defined formally (Sudkamp [1988]) but implemented to various degrees of formality. FSMs are an excellent process control technique. FSMs, as their name implies, track the *state* of a process, handling *events* which may cause the process to change state.

Formal FSMs are often represented as diagrams. Consider the example of a coin-operated candy dispenser whose FSM is shown in Figure 1.

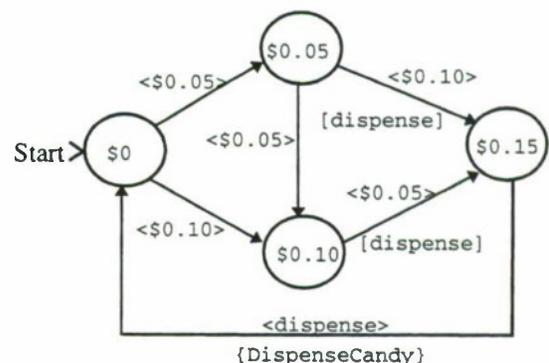


Figure 1: FSM for a coin-operated candy dispenser.

This machine accepts only nickels and dimes and dispenses candy worth \$0.15. In Figure 1, the circles represent states and arrows represent state transitions. Above the state transition line, in angle brackets (< >), is the event causing the transition. Square brackets ([]) represent any events that are generated as part of the transition. Braces ({ }) represent calls to STPs.

A formal FSM does not poll for events to change states; rather, events are generated and their arrival causes state transitions. This feature of formal FSMs is especially attractive because it eliminates inefficiencies introduced in polling.

IST considered implementing the DCA using ModSAF FSMs. In ModSAF, FSMs are not implemented formally: code is executed within states rather than by STPs during state transitions, and many state transitions are not event driven. Because formal FSMs provide an unambiguous way to control a process, they were used for the implementation

4.3 FSM Communication

An entity's cooperative behavior is implemented as an FSM. To cooperate, entities need to communicate and they do so via FSM communication. FSMs may communicate with each other (inter-FSM communication) or an FSM may communicate with itself (intra-FSM communication).

4.3.1 Inter-FSM communication

In inter-FSM communication, FSMs send events to each other. These events, called *external* events, often take the form of simulated "Radio Messages."

An entity may generate external events to itself. These events are generated from an Observation Module and are called Observation Events. Observation Events are generated in response to battlefield conditions.

4.3.2 Intra-FSM communication

FSMs communicate with themselves by sending *internal* events to themselves. Consider the FSM for the coin-operated candy dispenser shown in Figure 1. Assume that the FSM is in state "\$0.05." When a dime is deposited, the machine generates an internal event, *dispense*, to itself and transitions to the state "\$0.15." The receipt of the *dispense* internal event signals the FSM to transition to another state and execute an STP (*DispenseCandy*).

4.3.3 Event Queues

FSMs communicate by generating external events between themselves. When external events arrive, they are first mapped to internal events and then put into an event queue for processing. There are two approaches for handling external and internal events: use one or two event queues.

Using a common queue for external and internal events can lead to synchronization problems. State machine actions are non-preemptive; processing an internal event is done completely and, possibly, new internal events are generated in one execution thread. While internal events are being processed new external events may continue to arrive. If the external and internal events are processed in an interleaved manner unexpected situations can develop. Handling all possible interleaving of internal and external events is needlessly complicated.

The solution is to queue external and internal events in separate queues. External events are put into one or more *external event queues* while internal events are put into an *internal event queue*. No external event is dispatched until the internal event queue is empty. This allows all intra-machine communication (spawned by an external event) to complete without interference from new external events. The approach also allows a single external event to be re-mapped into several internal events. This reduces machine complexity and breaks complex external events into simpler requests.

4.4 FSM Engine

Because formal FSMs do not exist in ModSAF, an FSM Engine (Figure 2) was developed to run formal FSMs. The FSM Engine contains an FSM's description in a State-Event Table. The State-Event Table is created by reading a data file FSM description (Section 5.3). The table is indexed by a state/event pair that determines the new state of the FSM; the indices are the current state of the FSM and the internal event to be processed.

During the transition, external and internal events may be generated and STPs called.

The FSM Engine receives input from two sources: Signal PDUs and Observation Events. Signal PDUs contain radio messages and simulate radio communication. Observation Events are generated by the Observation Module in response to battlefield situations. Radio messages and Observation Events are external events which are queued on two separate queues: Radio Message Queue and Observation Event Queue.

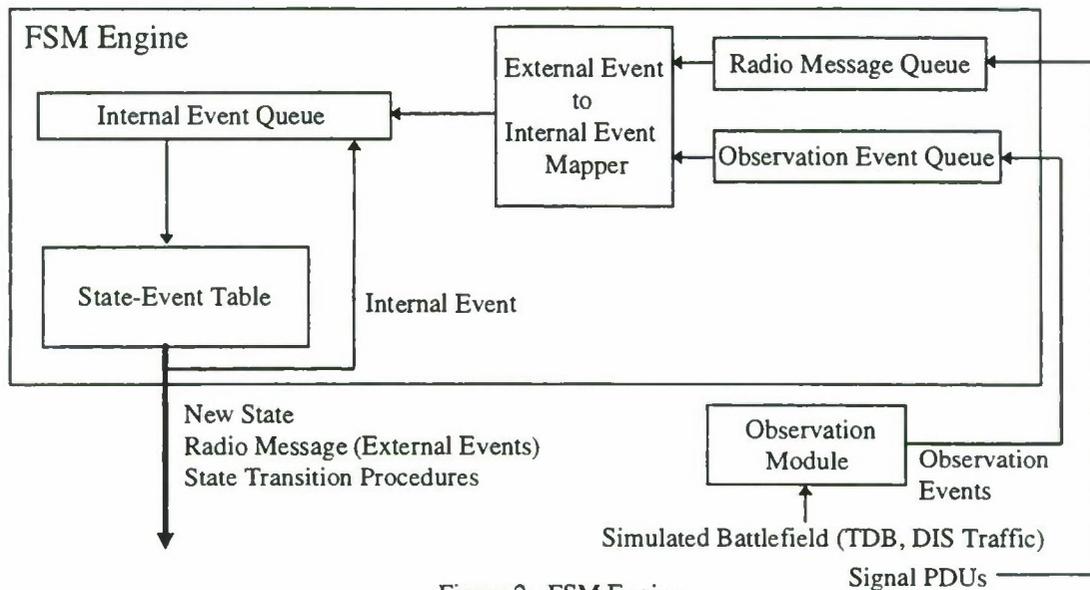


Figure 2: FSM Engine.

Periodically, the external event queues are checked to see if any external event is waiting to be processed.

The external event is removed from the queue, *mapped* into an internal event, and queued on the internal event queue. Then, internal events from the internal event queue are removed and processed.

To process events, the FSM Engine needs to be called periodically. This is done by calling the FSM Engine from a non-transitioning ModSAF FSM. Each time the ModSAF FSM becomes active, it calls the FSM Engine; the FSM Engine can be thought of as embedded within the ModSAF FSM. Note that the ModSAF FSM does not do anything. Its sole purpose is to ensure that the FSM Engine is called periodically; all the work required in processing events and changing the behaviors of entities is done by the FSM Engine.

5. Implementation

5.1 Hierarchy of Commanders

A vehicle can execute behaviors on many levels. Consider Vehicle 1 in Figure 3. The commander of this vehicle has three responsibilities, those of the Platoon Commander (PC), Section Commander (SC), and Vehicle Commander (VC). One way to represent the cooperative behavior of this commander would be to create a large and complex FSM that merges the platoon, section, and vehicle commander behaviors. This process can become arbitrarily complex as the hierarchy grows and commanders with more responsibilities are modeled.

For example, for the hierarchy shown in Figure 3, Platoon-Section-Vehicle Commander, Section-Vehicle Commander, and Vehicle Commander FSMs would be needed to encapsulate all classes of cooperative behaviors.

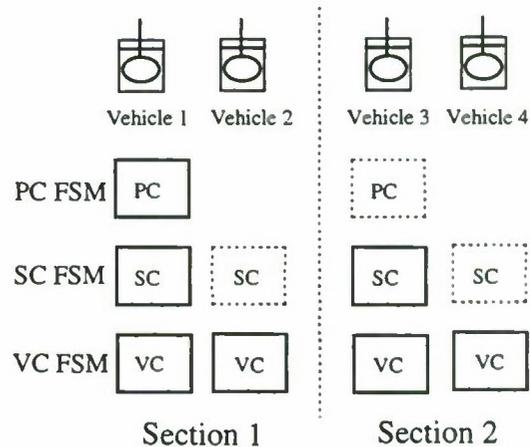


Figure 3: Hierarchy of Commanders.

Instead, IST established a hierarchy of commanders like the one shown in Figure 3. Each box in the figure represents a ModSAF FSM. Embedded inside each ModSAF FSM is the FSM Engine (Section 4.4).

This approach allows complex behaviors to be split into fundamental behaviors that are implemented as separate FSMs; complex FSMs containing merged behaviors are thus avoided. For example, Vehicle 1 (Figure 3) has three FSMs (Platoon, Section, and Vehicle Commander FSMs) controlling its behavior. Each FSM communicates with others.

The command hierarchy is created by higher level commander FSMs *spawning* lower level commander FSMs; for example, the Platoon Commander FSM spawns the Section Commander FSMs which in turn spawn Vehicle Commander FSMs.

In addition, there are next in command (*deputy*) commanders, shown by dotted boxes in Figure 3. Deputy commanders assume command when the original commanders are disabled so that the unit's mission can continue unhindered (Section 5.4). They model the behavior of the original commander but do not communicate with other entities. This allows them to continuously track the original commanders behavior and assume command in case the original commander is disabled. In Figure 3, Vehicle 3 is a deputy Platoon Commander (i.e., Platoon Sergeant), and Vehicles 2 and 4 are deputy Section Commanders for Vehicles 1 and 3 respectively.

To start, a user assigns a mission to the unit. As part of initialization, a data structure known as a **Role Matrix** is created. The Role Matrix is a two dimensional array of vehicle IDs and roles such as Platoon Commander, Section Commander, Vehicle Commander, and deputy commanders.

Vehicle ID	1	2	3	4
PC	1	0	0	0
PC deputy	0	0	1	0
SC	1	0	1	0
SC deputy	0	1	0	1
VC	1	1	1	1

Figure 4: Simplified Role Matrix.

Figure 4 shows a simplified Role Matrix for the commander hierarchy in Figure 3. The vehicles in the unit have IDs from 1 to 4. A "1" in a cell at the intersection of a vehicle ID column and role row means the vehicle is playing that role; for example, Vehicle 1 is the Platoon Commander, Section Commander, and Vehicle Commander. A "0" in a cell at the intersection of a vehicle ID column and role row means that the vehicle is not playing that role; for example, Vehicle 3 is not the Platoon Commander. Note that Vehicle 3 is a deputy Platoon Commander and Vehicles 2 and 4 are deputy Section Commanders. Because a Vehicle Commander's responsibility is limited to his vehicle's domain and another Vehicle Commander cannot assume his functions, there are no deputy Vehicle Commanders.

This is represented by the absence of a deputy Vehicle Commanders row in the Role Matrix.

A vehicle's Role Matrix is accessible from the vehicle's FSMs. Using the Role Matrix a vehicle can easily determine the role of other vehicles. In a real battlefield, a vehicle is designated roles before an exercise begins. The Role Matrix is a manifestation of this information in the computer.

5.2 Bounding Overwatch

The FSM architecture was tested on a platoon executing a Bounding Overwatch. Bounding Overwatch provides a simple and elegant way to test the architecture. In this behavior, a platoon advances by having its sections alternately move and overwatch the movement of the other section. The sections move until the platoon reaches an objective or enemy contact is made. By moving in this fashion, the platoon reduces the risk of being ambushed by enemy forces. The section that moves is called the Bounding Section while the section that keeps watch is called the Overwatch Section.

Sections 5.2.1 through 5.2.3 show FSMs for the Platoon, Section, and Vehicle Commanders for Bounding Overwatch. These FSMs communicate via radio messages (explicit cooperation). Note that in the following discussion overwatch is also called "Cover."

5.2.1 Platoon Commander FSM

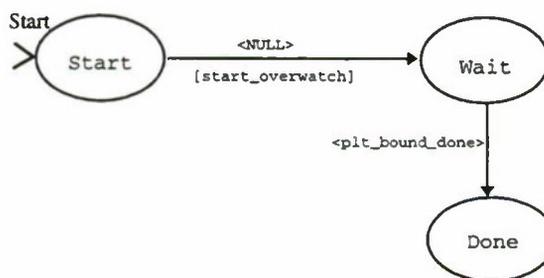


Figure 5: Bounding Overwatch Platoon Commander FSM.

To start the process, the Platoon Commander FSM sends a radio message, [start_overwatch], and transitions to the Wait state (Figure 5). It then stays there until it is informed (via event <plt_bound_done>) the platoon is at the objective, when it goes to the Done state.

5.2.2 Section Commander FSM

Figure 6 shows the Section Commander FSM for Bounding Overwatch. Initially the FSM is in the Waiting state. When a Section Commander receives the order to move (via event <start_move>), it sends a radio message, [start_move], to its Vehicle Commanders and transitions to the Move state. In this state the section moves toward an intermediate destination.

When a Section Commander receives the order to

The moving section may arrive at the intermediate destination in two ways. Either the Section Commander arrives first (event <my_bound_done> arrives) followed by the Wingman (event <wingman_complete> arrives) or vice versa. If the Section Commander arrives first, the FSM transitions to the I_arrive state. When the Wingman arrives (event <wingman_complete> arrives) the FSM transitions to the Cover state and as part of the transition does this: First, the Section Commander issues a radio message to its Vehicle Commanders to

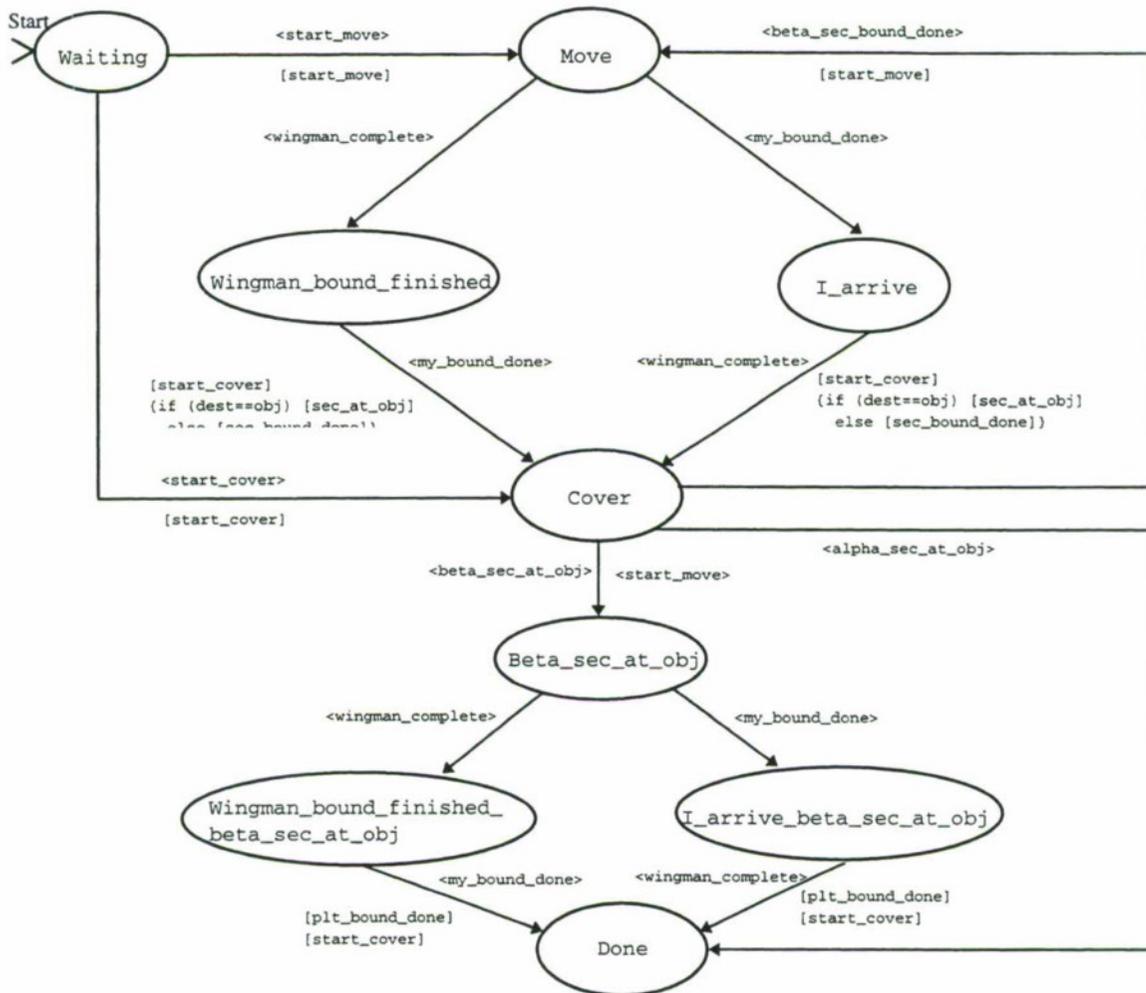


Figure 6: Bounding Overwatch Section Commander FSM.

cover (via event <start_cover>), it sends a radio message, [start_cover], to its Vehicle Commanders and transitions to the Cover state where it overwatches the moving section.

start cover and second, checks if the intermediate destination is the objective. If the section is at the objective it sends a radio message, [sec_at_obj] (section at objective), otherwise the section is at an intermediate destination and a radio message, [sec_bound_done] (section bound done), is sent.

The section now overwatches the movement of the other section, which has transitioned from overwatch to move.

5.2.3 Vehicle Commander FSM

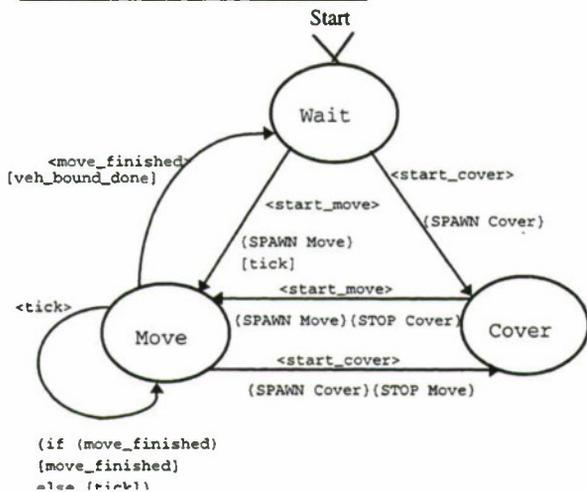


Figure 7: Bounding Overwatch Vehicle Commander FSM.

Figure 7 shows the Bounding Overwatch Vehicle Commander FSM. Initially, the FSM is in the Wait state. The order to start a move (via event `<start_move>`) takes the FSM to the Move state. As part of the transition the FSM spawns a ModSAF Move task (via STP `{SPAWN Move}`). This is a low level ModSAF behavior that a vehicle uses to travel.

Periodically, the Vehicle Commander checks if it has finished traveling. This check is made every time the FSM receives a `<tick>` event. When the move is finished the Vehicle Commander sends a radio message, `[veh_bound_done]` (vehicle bound done), and transitions to the Wait state to receive further orders from its Section Commander.

5.3 Describing Commander FSMs in Data Files

FSMs describing the cooperative behavior of commanders are written in data files. This approach allows quick behavior specification; a user only needs to change a data file to create a new behavior, code changes are not required.

5.3.1 FSM Grammar Production Rules

To describe FSMs, production rules were developed. These production rules specify the structure of an FSM description. FSM descriptions are "parsed"

based on production rules and a representation of the FSM is created inside the computer.

The production rules for the FSM grammar are:

```

FSM  => (State)
State => (state_name (Event)) ||
         (state_name (Event)) State
Event => (event_name next_state
         (STP)) ||
         (event_name next_state (STP))
         Event
STP  => (TRUE (Actions)) STP ||
         (PRED (Actions)
         (Actions)) STP || (FUNC) || ε
Actions => MSG string Actions ||
           EVENT string Actions ||
           SPAWN string Actions ||
           STOP string Actions || ε
PRED  => string
FUNC  => string

```

where:

ϵ is the symbol for a NULL string.

An operator (TRUE, PRED, and FUNC) specifies how Actions are to be treated. TRUE means execute unconditionally the Actions that follow. PRED is a user specified *predicate function*. Based on the result of the predicate function, true or false, the first or the second list of Actions is executed. FUNC is a user defined function. The Actions specify what is to be done. MSG means to broadcast the string that follows as a radio message. EVENT means to put an internal event, string, on the internal event queue for processing. SPAWN means to spawn a ModSAF task specified by string. STOP means to stop a ModSAF task, specified by string, which was spawned earlier.

5.4 Change in Command

In the real world, when a commander becomes disabled, the next in command (deputy) commander takes charge. Shifting command enable units to continue their missions with minimal disruption. This important real world feature was implemented in this project.

In the simulated battlefield, a deputy commander models his commander's cooperative behavior via an FSM similar to the commander's FSM. This model (FSM) is constantly updated through receipt of observation and radio messages. This information keeps the model synchronized with the original commander's cooperative behavior. A deputy

commander "knows" what his commander is doing because the deputy's commander's FSM goes through the same transitions as his commander's FSM. (An important characteristic of the model is that information flow is *unidirectional*; i.e., information contained in observation and radio messages flows into the model but does not flow out, e.g., a deputy commander does not transmit radio messages which are intended for transmission by the original commander). If required, a deputy commander can assume command and continue the mission from the last executed command of the original commander.

In the simulated battlefield, all entities watch out for each other and respond when someone is disabled. When an entity is disabled, such as by a firepower kill, another Vehicle Commander is notified by a "vehicle destroyed" *observation message* from its Observation Module. The observation message contains the Vehicle ID of the disabled vehicle. Upon receipt of the Observation Message, the Vehicle Commander sends a "vehicle destroyed" *radio message* containing the disabled vehicle's Vehicle ID. This message is sent only once.

A deputy commander runs a ModSAF FSM, called a Monitor FSM, to process vehicle-destroyed messages. Because deputy commanders are present at different levels in the command hierarchy, such as deputy commanders for Platoon and Section Commanders (Section 5.1), Monitor FSMs are also present at different levels. When a Monitor FSM receives a vehicle-destroyed message it checks the vehicle ID in the message with the vehicle ID of the original commander. If they are different, the message is discarded. Otherwise, the Monitor FSM changes the Role Matrix (Section 5.1) to reflect the change of command.

When a commander is disabled, ModSAF designates another entity as the commander and restarts the unit's mission. ModSAF developers believe that restarting the mission reflects the change in command; another entity is "promoted" to the commander. ModSAF's internal architecture imposed a barrier to implementing this change of command process. The new commander plans and executes the task using the current vehicles and positions.

Because changes to the ModSAF software, to disable automatic mission restart, involve a fundamental change to the ModSAF architecture, IST did not pursue this approach. However, to test the transfer of command, IST designated an entity to be the Platoon Commander which is different than the ModSAF-

designated Platoon Commander. When the IST-designated Platoon Commander is destroyed, control is transferred to the Platoon Sergeant.

6. Results

The formal FSM DCA was implemented in ModSAF version 1.5.1. Bounding Overwatch with explicit and implicit cooperation was implemented and tested.

When the Bounding Overwatch order was given to a platoon using explicit cooperation, both vehicles in the Bounding Section started simultaneously because they each received the order via a radio message. When they reached the first overwatch position, the vehicles stopped and the Overwatch Section started moving. The sections repeated this process until the platoon was at the objective.

When the Bounding Overwatch order was given to a platoon using implicit cooperation, the Section Commander of the Bounding Section started first. His Wingman remained stationary for a while until he noticed the Section Commander's movement. The Wingman then followed the Section Commander to the first overwatch position. When the Bounding Section stopped, the Overwatch Section started its bound. IST noted that the vehicles in the Bounding Section did not maintain as tight a formation, as compared to the formation of the vehicles in the Bounding Section in the explicit cooperation Bounding Overwatch, because of observation delays.

7. Conclusions

This project has implemented a Decentralized Control Architecture (DCA) within the ModSAF CGF system. In addition to mirroring cooperation in the real world, DCAs allow cooperation within larger units (companies, battalions, etc.) to be modeled with little increase in complexity. Explicit and implicit cooperation between entities has been demonstrated within a platoon engaged in a Bounding Overwatch.

The cooperative behavior of an entity is implemented through FSMs. An entity's cooperative behavior is described in data files. These descriptions are read and converted into FSM representations inside the computer. Communication between entities is implemented by FSM communication. FSMs communicate by sending each other external events implemented as radio messages. An FSM communicates with itself by sending internal events. An FSM Engine, embedded within ModSAF, "reads" the description and executes the defined behavior.

The FSM Engine is general purpose and can be used by other ModSAF code; ModSAF has been extended.

The FSMs use low level ModSAF behaviors. For example, a ModSAF task is used for vehicle travel as the underlying fundamental behavior. This attempts to reuse code as much as possible. Thus, in addition to being extendible, the approach is built on top of ModSAF.

Simpler implementations result as a consequence of the FSM approach. Instead of modeling various responsibilities of a commander as a large and complex FSM, responsibilities corresponding to different levels in the command hierarchy are modeled as separate FSMs which communicate with each other.

8. References

- Corkill, Daniel (1991). "Blackboard Systems". *AI Expert* 6(9):40-47, September 1991.
- Dai, H., Hughes, J. G. and Bell, D. A. (1993). "A Distributed Real-Time Knowledge-Based System and its Implementation using Object-Oriented Techniques". *Proceedings of the International Conference on Intelligent and Cooperative Information Systems*, May 1993, pp. 23-30.
- Decker, K. S. (1987). "Distributed Problem-Solving Techniques: A Survey". *Proceedings of the IEEE Transactions on Systems, Man, and Cybernetics*, Vol. SMC-17, No. 5, September/October 1987.
- Fisher, M. and Woodridge, M. (1994). "Specifying and Executing Protocols for Cooperative Action". *CKBS-94, Proceedings of the Second International Working Conference on Cooperating Knowledge Systems*, Springer-Verlag, 1994.
- Harmon, S. Y., Aviles, W. A., and Gage, D. E. (1986). "A Technique for Coordinating Autonomous Robots." *IEEE International Conference on Robotics and Animation*, 1986, Vol. 1, page 666.
- Laird, John E., Jones, Randolph M., and Nielsen, Paul E (1994). "Coordinated Behavior of Computer Generated Forces in TacAir-Soar", *Proceedings of the Fourth Conference on Computer Generated Forces and Behavioral Representation*, University of Central Florida, Orlando, Florida, pp. 325-332.
- Laengle, T. and Lueth, T.C. (1994a). "Decentralized Control of Distributed Intelligent Robots and Subsystems". *Proceedings of the IFAC Symposium on Artificial Intelligence in Real Time Control (AIRTC '94)*.
- Laengle, T. and Lueth, T.C. (1994b). "Task Description, Decomposition, and Allocation in a Distributed Autonomous Multi-Agent Robot System". *Proceedings of the 1994 IEEE/RSJ International Conference on Intelligent Robots and Systems*.
- Lefebvre, D. R. and Saridis, G. N. (1992). "A Computer Architecture for Intelligent Machines". *Proceedings of the 1992 IEEE International Conference on Robotics and Automation*.
- Loral (1995a). "Libuoverwatchmove Online Documentation", Loral Advanced Distribution Simulation, Cambridge, Massachusetts, April 28, 1995.
- Lun, V. and MacLeod, I. M. (1992). "Strategies for Real-Time Dialogue and Interaction in Multiagent Systems". *Proceedings of the IEEE Transactions on Systems, Man and Cybernetics*, Vol. 22, No. 4, July/August 1992.
- Noreils, Fabrice R. (1992a). "An Architecture for Cooperative and Autonomous Mobile Robots". *Proceedings of the 1992 IEEE International Conference on Robotics and Automation*, Nice, France, May 1992, pp. 2703-2710.
- Noreils, Fabrice R. (1992b). "Multi-Robot Coordination for Battlefield Strategies". *Proceedings of the 1992 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Raleigh, NC, July 7-10, 1992, pp. 1777-1784.
- Noreils, Fabrice R. (1992c). "Coordinated Protocols: An Approach to Formalize Coordination Between Mobile Robots". *Proceedings of the 1992 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Raleigh, NC, July 7-10, 1992, pp. 717-724.
- Noreils, Fabrice R. (1993). "Toward a Robot Architecture Integrating Cooperation between Mobile Robots". *The International Journal of Robotics Research*, vol. 12, no. 1, February 1993.
- Occello, M. and Demazeau, Y. (1994). "Building Real Time Agents using Parallel Blackboards and its use for Mobile Robotics". *Proceedings of the 1994 IEEE International Conference on Systems, Man and Cybernetics*, San Antonio, October 1994.
- Ohko, T., Hiraki, K, and Anzai, Y. (1993). "LEMMING: A Learning System for Multi-Robot Environments". *Proceedings of the 1993 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Vol. 2, July 1993, pp. 1141-1146.

- Parker, Lynne E. (1994). "ALLIANCE: An Architecture for Fault Tolerant, Cooperative Control of Heterogeneous Mobile Robots". Proceedings of the IEEE/RSG/GI International Conference on Intelligent Robots and Systems (IROS '94) Vol. 2, 1994, pp. 776-783.
- Payton, David W. and Dolan, Charles P. (1991). "Cooperative Control". Seminars on Robotics in the Air/Land Battlefield, NATO Defense Group, 1991.
- Rajput, S. and Karr C. R. (1995). "Cooperative Behavior in ModSAF," Contract Report IST-CR-95-35, Institute for Simulation and Training, University of Central Florida.
- Shin, Kang G. and Epstein, Mark E. (1990). "Intertask Communications in an Integrated Multirobot System". Multirobot Systems, IEEE Computer Society Press, Los Alamitos, California, 1990.
- Shoham Y., and Tennenholtz M (1992). "On The Synthesis Of Useful Social Laws For Artificial Agents Societies", (preliminary report), *Proceedings of AAAI-92*, Morgan Kaufmann, 1992.
- Smith, R. G. and Davis, R. (1981). "Framework for Cooperation in Distributed Problem Solving". Proceedings of the IEEE Transactions on System, Man and Cybernetics, Vol. SMC-11, No. 1, January 1981, pp. 61-70.
- Sudkamp, Thomas A. (1988). Languages and Machines: An Introduction to the Theory of Computer Science. Addison-Wesley Publishing Company Inc., 1988.
- Tambe, Milind, and Rosenbloom, Paul S (1995). "Agent Tracking in Complex Multi-agent Environments: New Results", *Proceedings of the Fifth Conference on Computer Generated Forces and Behavioral Representation*, University of Central Florida, Orlando, Florida, pp. 125-133.
- US Army (1990). "FM 7-7J: The Mechanized Infantry Platoon And Squad (Bradley)", Coordinating Draft, Department of the Army, United States Army Infantry School, Fort Benning, Georgia 31905.
- Wang, J. (1994). "On Sign-board Based Inter-Robot Communication in Distributed Robotic Systems". Proceedings of the 1994 IEEE International Conference on Robotics and Automation, Vol. 2, May 1994, pp. 1045-1050.

9. Acknowledgment

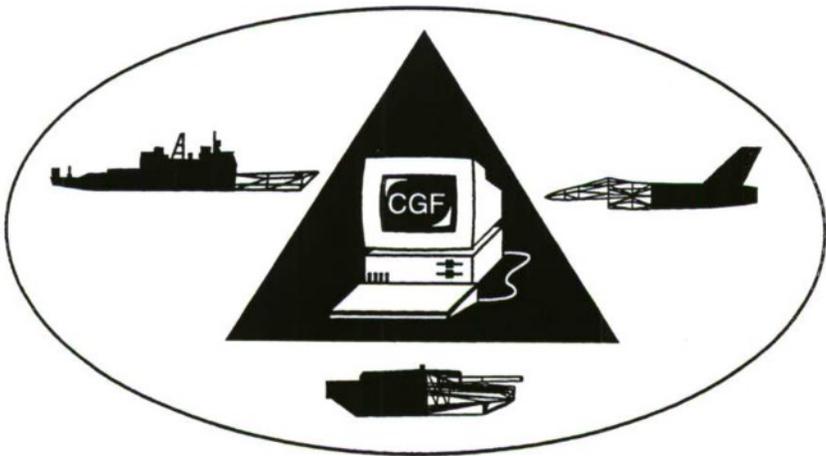
This research was sponsored by the US Army Simulation, Training, and Instrumentation Command

as part of the Intelligent Simulated Forces project, contract N61339-92-C-0045. That support is gratefully acknowledged.

10. Authors' biographies

Sumeet Rajput is an Associate Computer Scientist in the Intelligent Simulated Forces project at the Institute for Simulation and Training. Mr. Rajput has a Master of Science degree in Computer Science from the University of Central Florida and is an MBA student at the University of Central Florida. His research interests are in the areas of Computational Geometry, Physical Modeling, and Computer Generated Forces.

Clark R. Karr is a Program Manager and the Principal Investigator of the Intelligent Simulated Forces project at the Institute for Simulation and Training. Mr. Karr has a Master of Science degree in Computer Science. His research interests are in the areas of Artificial Intelligence and Computer Generated Forces.



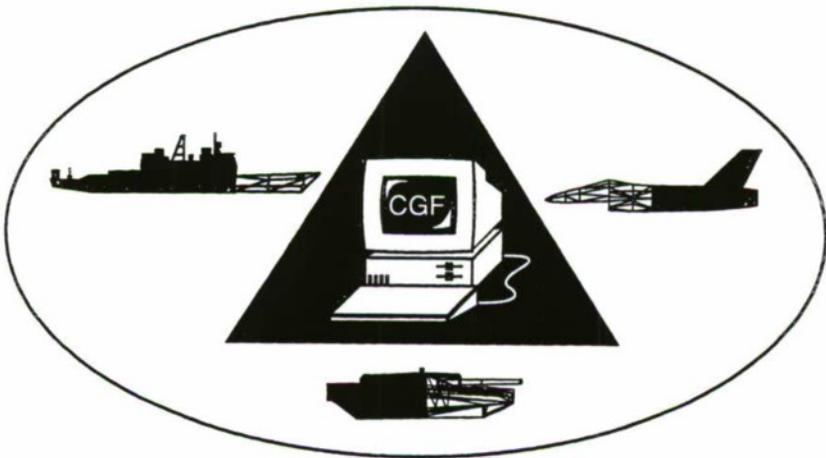
Session 3b: Exercise Planning - AAR

Coulter, University of Michigan

Juliano, SAIC

Pittman, Oregon Graduate Inst.

Pratt, NPGS



A Briefing-Based Graphical Interface for Exercise Specification

Karen J. Coulter and John E. Laird
Artificial Intelligence Laboratory
The University of Michigan
1101 Beal Ave.
Ann Arbor, MI 48109-2110
{kcoulter, laird}@umich.edu

1. Abstract

One of the most time-consuming aspects of using computer generated forces (CGF's) is specifying their missions. At best, current tools organize information around the individual entity or small groups of entities. At worst, current tools organize information around the simulator code implementation. It is left to the user to translate between actual military command structure and current mission specification formats for CGF's, and to create higher levels of organization, possibly duplicating information shared at lower command levels. As we attempt to include more and more entities within a scenario, mission specification can become the most difficult part of running an exercise. To ease the burden on the user, we have created a graphical tool that allows users to specify complete exercises for Navy fixed-wing aircraft missions in a manner consistent with military command structure. This tool was used successfully to configure agents for a simulation exercise in October 1995, and more recently was used to great extent by novices during a week-long tutorial on our TacAir-Soar synthetic agents.

2. Introduction

In order to achieve human-like performance in simulations, intelligent synthetic agents require the same input parameters that are briefed to actual pilots for their missions. Depending on the scope of the exercise and the mission type, the number of input parameters which must be specified for each agent can range from 50 to 100 or more. Of these, often less than 10 parameters are unique to an individual agent. The rest of the input data is common to other agents in the same mission or event. However, existing tools for generating synthetic forces require users to specify param-

eters on an agent by agent basis, requiring a huge duplication of effort when more than one agent is defined. These tools also present the user with the list of all parameters required for all possible mission types, leaving it up to the user to determine which are relevant to the particular mission type currently selected. Because these tools support the configuration of so many different types of synthetic forces in a very generic way, there is very little error checking available, leaving the user to track down problems at simulation time rather than during data entry.

In developing a new tool for configuring our pilot agents, our goals were to reduce the time and effort required to configure the agents for large exercises, and to allow domain experts to create missions without having to understand the underlying implementation of the simulation. In order to achieve these goals and avoid the problems of existing tools, the TacAir-Soar Exercise Editor was developed according to the following design criteria:

- The Exercise Editor should be organized in a hierarchical fashion, following the briefing structure used by the Navy. Agents defined at the lowest level inherit all data values entered at higher levels.
- The information presented on the screen should be data-driven, so that the value of certain parameters will affect whether or not the user will be prompted to enter values for other parameters.
- The potential for user-input error should be reduced by giving the user lists of possible settings from which to choose, by providing reasonable default values, and by implementing error-checking of data as it is being entered by the user.

The remainder of this paper describes the organization and implementation of the Exercise Editor and discusses the progress and results to date, and the future work planned. The Exercise Editor has been implemented primarily to support intelligent forces implemented in the TacAir-Soar system (Tambe et al., 1995 and Laird et al., 1995), although we expect it to be easily adaptable to many types of synthetic forces.

3. Organization

3.1 Briefing Structure

The TacAir-Soar Exercise Editor organizational structure is based on actual pilot briefing hierarchy as described to us by former Navy pilots during knowledge acquisition sessions (Petersen, 1995 and Checchio, 1995-96). There are four briefing levels in the editor hierarchy: the Exercise level, Event level, Mission level and Element level. The Exercise level includes general information that is expected to be relevant to all pilot agents during the whole exercise, for instance rules of engagement, climatology, and terrain data. An Exercise consists of one or more Events, and each Event includes information relevant to all activities which occur during a specific time period, for instance launch and recovery information and weather reports. Each Event consists of one or more Missions, a coordinated activity designed around a specific target or objective. At the Mission level, items such as the specific mission type are identified, route and target parameters are defined, the likelihood of ground and air threats are indicated and controller agencies are identified. Finally, at the Element level, individual aircraft are assigned, and call-signs, radio frequencies and formations are determined. Individual pilot agents defined at the Element level have knowledge of all data within their own Mission specification and at the Event and Exercise levels above it.

The Exercise Editor is used to define one exercise at a time. Below the Exercise level, there can be any number of Events, Missions and Elements. Events and Missions are identified by their respective Event and Mission numbers. Agents at the Element Level are identified by call-sign. The only order imposed on the lower levels is that the Events are numbered sequentially as they are created. This is done mainly to support the directory structure for the editor input and output files, but also follows the protocol of the Navy briefing hier-

archy. Mission numbers are assigned by the user as are agent call-signs. At the Element level, the user can create an individual vehicle, a section (2 aircraft) or a division (3 or 4 aircraft) to carry out the mission.

Much of the information briefed to pilots is dependent on the type of mission being carried out and by the number and type of aircraft constituting an element. Parameters required by a single agent flying a barrier combat air patrol are quite different from the parameters required by a division of aircraft conducting a strategic attack. The Exercise Editor prompts the user for only the parameters required for a particular mission type and element configuration. All parameters related to a specific mission type are organized into a form which, whenever possible, follows the same format used by Navy pilots. When the user selects a mission type, the corresponding form is presented for input. Likewise, if an element consists of only a single vehicle, the user will not be asked to define the formation type; and the list of possible formation types displayed will be different for a section than for a division of aircraft. Users are not required to decide which parameters must be defined for correct mission performance, since those that are not required will not be presented to the user for input.

3.2 Graphical Structure

The graphical structure of the Exercise Editor parallels the briefing structure, resembling an inverted tree, with the Exercise level at the top and the Elements making up the terminal leaf nodes. Each level has its own distinct screen format identifying the current level, and providing pushbuttons to move up or down to the next level. The user interacts with only one node at a time.

At each level, the user is presented with a simple and consistent set of widgets for specifying parameters. An example of the Exercise level screen is shown in Figure 1. The five basic widgets used are:

- **Buttons:** Used to navigate from one level to another and to pop up/down related screens.
- **OptionMenus:** Displays the current setting for a parameter. Options which are not currently selected are only displayed when the menu is activated.
- **PullDownMenus:** Used for various functions, such as selecting a particular mission to dis-

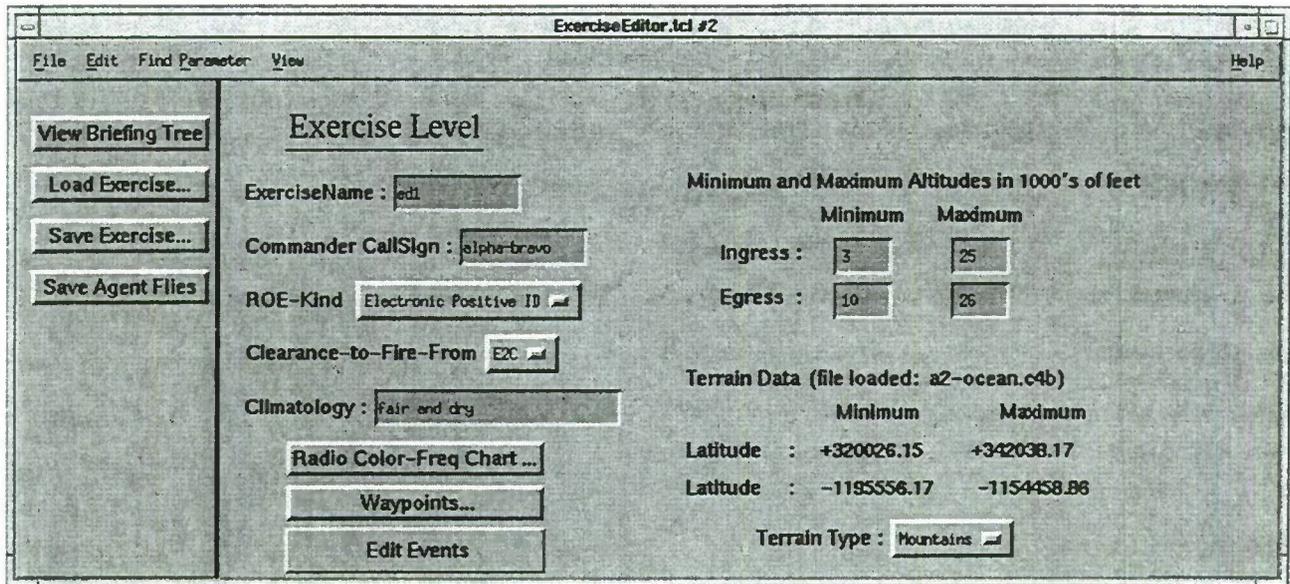


Figure 1: Sample Exercise Level Screen

play, as indicated by the label on each Pull-down-Menu.

- **TextEntry:** Used to enter free-form text values.
- **RadioButtons:** Displays the current setting for a parameter, but also displays all values not currently selected.

Wherever possible, the user is presented with a list of possible choices to select from, rather than being forced to enter a new value at the keyboard, which is more time-consuming and more prone to user error. Free-form text entry widgets are used only when necessary, such as for specifying mission numbers and agent call-signs.

In order to simplify the amount of information the user must interact with at one time, sets of related input parameters are grouped into forms, which are displayed by activating buttons on the main screen. For example, at the Exercise level, there are usually more than a dozen waypoints specified, with the name, latitude and longitude required for each point. Rather than try to display them all with the rest of the Exercise level data, the waypoints are grouped into a single form that is popped up by pressing the "Waypoints..." button at the Exercise level (see Figure 1). Having a separate form for the waypoints also allows for the waypoint data to remain visible while the user traverses Event, Mission and Element levels. The grouping of related parameters is used throughout the Editor, and allows for simple, consistent

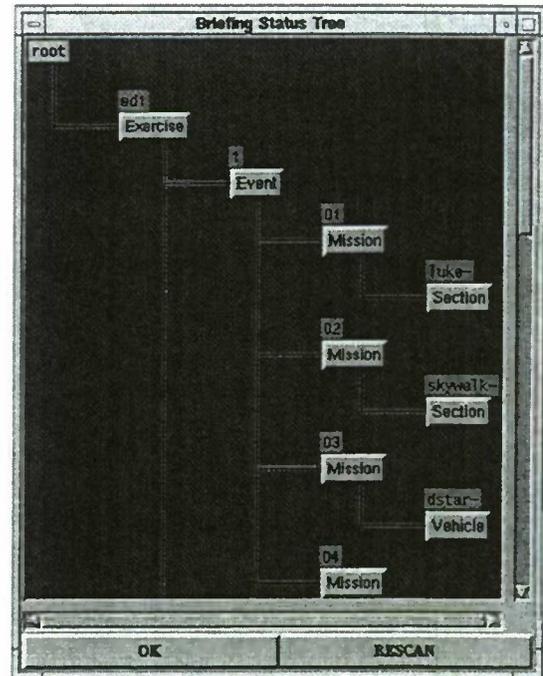


Figure 2: Briefing Status Tree

screens through the four main levels.

Several features have been built in to aid the user in navigating and reviewing the events and missions which constitute an exercise. The Briefing Status Tree shown in Figure 2 is a separate window that provides a summary view of all Events, Missions and Elements defined in the Exercise. The

Event and Mission numbers are identified, and call-signs and the number of vehicles are displayed for each Element. Each object in the Briefing Status Tree is represented by a pushbutton, which when activated will cause that object to be displayed in the main window. The Briefing Status Tree allows users to quickly traverse the nodes in the exercise in any order; users are not limited to moving up and down the connected nodes of the tree. At the Event level, a more detailed summary of all missions defined in that event is presented in a format used by the Navy, listing aircraft type, mission type, number of aircraft per mission, station name and controller call-sign.

A limited amount of error checking has been implemented, which alerts the user to inconsistencies at data entry, rather than forcing the user to track them down at execution time. Users are immediately alerted at the Mission level when controller call-signs are specified for which no controller agent has been defined, and are never permitted to enter non-numeric values where integers are required.

4. Implementation

The Exercise Editor is implemented in Tcl, the Tool Command Language, developed by John Ousterhout, currently with Sun Microsystems. Tcl is an interpreted scripting language with several hundred extensions, available in the public domain. Two of these extensions were used in creating the Exercise Editor to enhance its capabilities. The [incr Tcl] extension package, written by Michael McLennan at AT&T Bell Laboratories, adds object-oriented facilities to Tcl, and provides a means of supporting inheritance and encapsulating data in the Exercise Editor. Each level of the Editor has its own associated object type and methods for handling the data and propagating dependencies. Functionality at each level is easily expanded by changing the definition of the object and its methods, without compromising the integrity of the other levels. The other extension package used for the Exercise Editor is tclMotif, developed by Jan Newmarch at the University of Canberra, which allows Tcl programs to use the Motif set of widgets to create a graphical interface. tclMotif does not duplicate the Motif widget set — it uses the resident Tm library to create and manipulate the widgets. Thus the Exercise Editor graphical interface is consistent with all Motif ap-

plications on a user's system and does not require the user to learn an additional set of graphical behaviors.

The battlefield simulation and aircraft used in TacAir-Soar are provided by ModSAF (Calder et al., 1993). In order for the Exercise Editor to generate ModSAF scenario files for running the simulations, several ModSAF libraries are linked into the Editor. The Editor then makes calls to ModSAF routines to read terrain database information, and to create the vehicles and waypoints in the persistent object database and save the database out to a scenario file. No modification to the ModSAF libraries is necessary and the function calls are straightforward.

In addition to generating ModSAF scenario files and reading terrain database files, the Exercise Editor generates agent mission files and exercise data files. Each agent mission file contains all data required by a particular agent to carry out its mission; they are read by TacAir-Soar at startup. The exercise data files store the complete data for the entire exercise and are used only by the Exercise Editor for saving and restoring exercises.

5. Progress and Results

Our goal in creating the TacAir-Soar Exercise Editor is to reduce significantly the time and expertise required to configure exercises for TacAir-Soar agents. A prototype of the Exercise Editor was used to configure agents for a simulation exercise called ED-1 in October 1995. Although the prototype required several intermediate steps in order to start the simulation, the amount of time required to configure all of the agents was reduced from days to hours. The current version of the Exercise Editor generates the simulation scenario files directly, allowing a user to create complex events with many missions in just a few hours. Using the Exercise Editor makes it much easier to make changes to the exercise parameters and propagate those changes to all agents. It is now being used to generate all new missions for simulation exercises. In April, during a week-long tutorial on the TacAir-Soar system, novices were given a one-hour presentation on the Exercise Editor, followed by a short, hands-on working session. The users were able to modify existing exercises with few errors, and were confident that they would be able to generate an entire exercise on their own with little dif-

ficulty. The Exercise Editor was used throughout the week to generate scenarios for testing various aspects of the TacAir-Soar system, allowing users to spend more time learning about the capabilities of the synthetic agents and less time figuring out how to specify missions.

6. Future Work

As we continue to expand the capabilities of our synthetic agents, the Exercise Editor must be modified to support new agent input requirements, which will consume most of our development efforts for the near term. Work is currently underway to investigate adding a map-based interface to the Editor to allow users to specify flight plans and waypoints graphically, rather than through text input. Other proposed enhancements to the Editor include adding a copy feature to duplicate missions within an exercise, implementing better help facilities, providing a mechanism to specify which agents should run on which workstations, and adding a graphical capability for configuring weapons loadouts on individual aircraft.

7. Acknowledgements

This research was supported at the University of Michigan as part of contract N00014-92-K-2015 from the Advanced Systems Technology Office of the Defense Advanced Research Projects Agency and the Naval Research Laboratory. The research presented here has benefited greatly from the efforts of Randy Jones, Frank Koss, Paul Nielsen, BMH Associates, Inc., and the other members of the Soar/IFOR project.

8. References

- Calder, R.B., Smith, J.E., Courtemanche, A.J., Mar, J.M.F., and Ceranowicz, A.Z. (1993). ModSAF Behavior Simulation and Control. In *Proceedings of the Third Conference on Computer-Generated Forces and Behavioral Representation*, (pp. 347-356). Orlando, FL.
- Checchio, M., BMH, Associates, Inc., personal communications, 1995-96.
- Laird, J. E., Johnson, W. L., Jones, R. M., Koss, F., Lehman, J. F., Nielsen, P. E., Rosenbloom, P. S., Rubinoff, R., Schwamb, K. B., Tambe, M., Van Dyke, J., van Lent, M., & Wray, R.

E. (1995). Simulated Intelligent Forces for Air: The Soar/IFOR Project 1995. In *Proceedings of the Fifth Conference on Computer-Generated Forces and Behavioral Representation*, (pp. 27-36). Orlando, FL.

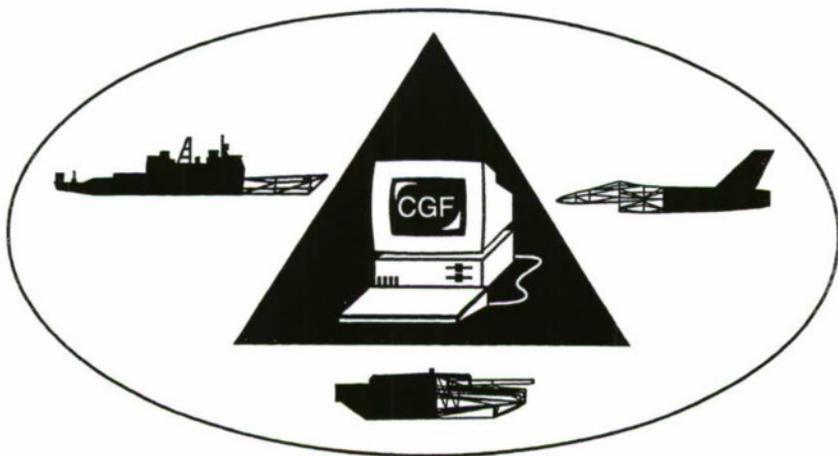
Petersen, C., BMH, Associates, Inc., personal communications, March 1995.

Tambe, M., Johnson, W. L., Jones, R. M., Koss, F., Laird, J. E., Rosenbloom, P. S., & Schwamb, K. B. (1995). Intelligent agents for interactive simulation environments. *AI Magazine*, 16(1), 15-39.

9. Biographies

Karen J. Coulter received her B.S. in Physics from the University of Michigan in 1984 and her M.S. in Computer Science from the Illinois Institute of Technology in 1993. She is currently a systems research programmer in the Artificial Intelligence Laboratory at the University of Michigan, where she is enhancing the Soar infrastructure and developing tools for the Soar/IFOR project. Her main areas of interest are in human-computer interactions.

John E. Laird is an associate professor of Electrical Engineering and Computer Science and the director of the Artificial Intelligence Laboratory at the University of Michigan. He received his B.S. degree in Computer and Communication Sciences from the University of Michigan in 1975 and his M.S. and Ph.D. degrees in Computer Science from Carnegie Mellon University in 1978 and 1983, respectively. His interests are centered on creating integrated intelligent agents (using the Soar architecture), leading to research in problem solving, complex behavior representation, machine learning, and cognitive modeling.



Scenario and Infrastructure Analysis to Measure Large-Scale CGF Exercise Performance

Michael Juliano, Robert D'Urso, Dr. Ben Wise
SAIC

20 Burlington Mall Rd
Burlington MA. 01803

mjuliano@bos.saic.com, rdurso@bos.saic.com, bwise@bos.saic.com

Dr. Edward Powell
SAIC

Suite 1100
1100 N. Glebe Road
Arlington VA. 22201
epowell@stow.std.saic.com

1. Abstract

The Scenario Analysis and Infrastructure Analysis tools (SAT/IAT) are currently being developed by SAIC under the SEID contract to support the STOW (Synthetic Theater of War) Exercise Implementation (XI) system. The simulation is responsible for aiding in the exercise generation, management, and infrastructure design and testing for Computer Generated Forces (CGF). Since future CGF exercises, including STOW, are required to support a distributed exercise of a large number of entities, a pre-exercise faster than real-time determination of scenario, network and computational validity is necessary. The SAT/IAT simulation executes at a rate of hundreds to one real-time. The modeling of an entire week's exercise in less than an hour allows a user to try different configurations with a quick turnaround time.

One area in which the SAT/IAT is useful is in generating performance numbers for CGF applications before execution of a full scale exercise. The IAT can predict loads based upon simple models of host processor capabilities. Each CGF application's computational resources are calculated as the exercise unfolds. Predictions of network traffic at both the Wide Area Network (WAN) and Local Area Network (LAN) are provided. Therefore, for a given scenario, the IAT can generate information such as the number of entities that can be modeled per simulation process per host. Also, it is possible to provide the number of remote packets received per host, the number of packets received per LAN, and

the number of packets transmitted over the WAN as a function of time.

Another area where the SAT/IAT is useful is in predicting entity migration via dynamic load balancing. Using host processor loads over time, the IAT can predict when CGF applications are most likely to migrate entities to other CGF applications. The effects of entity migration upon the network infrastructure will be provided. Various algorithms that implement load balancing and entity migration can be modeled in order to determine the optimal computational and network performance.

The major objective of the SAT/IAT is to simulate a given scenario at a very coarse level and provide processing and network loads upon a given infrastructure and network topology based upon the SAT output. The goals of the SAT are to 1). validate laydown information of an exercise's units on a synthetic environment (SE) database, 2). provide unit location, strength, logistical consumption, attrition, rate of sensor detection's and losses during execution of the scenario, 3). generate a profile of the units' processing and network requirements based upon their activity, and 4). provide coarse level network traffic to the IAT. The goals of the IAT are to 1). map the respective units and their entities to simulation processes to computers and the network topology, 2). calculate the processing and network loads on the infrastructure based upon the SAT output, and 3). supply the user with time-based predictions of network and computationally utilization. This includes an indication of overloading of resources to allow the user to

reconfigure the hosts, network, and mapping units to nodes and simulation sites.

The current process for Infrastructure and Scenario analysis involves a great deal of manual data manipulation. This process tends to be time consuming and less accurate. The SAT/IAT tool functionality will greatly reduce the time for analysis and increase the validity of the resultant analysis data.

2. Scenario Analysis Tool

2.1 SAT Functionality

The goal of the Scenario Analysis Tool is to provide exercise planners with the capability to design, execute and refine a given exercise scenario. The SAT allows the user to initialize, save, load and perform analysis of a given scenario at a faster than real time execution rate. The quick turnaround allows the user to define a scenario at a very high level and refine it to a level of detail to be used as a starting point to plan an exercise. The SAT simulation performs analysis of a scenario at a coarse level of execution and is not intended to provide or predict battle outcomes. Instead it is used to determine if a given scenario is realistic in the sense of scenario planning and to interface with the IAT to predict and optimize network loading for a selected topology.

The user begins by creating an exercise to analyze. The Corps Level Computer Generated Forces (CLCGF) Simulation Interface Unit (SIU) is utilized by the user for exercise laydown, initialization and visualization. Once the user defines the exercise that is under analysis, the data is saved in the SIU format and loaded to the SAT. Units represented at various echelons are placed on the SE database. The units are assigned routes and activities. The activities currently simulated are moves, attacks, and radar scans. The data is reformatted and sent to the SAT for analysis. The unit is assigned missions and is assigned to a simulation process. An algorithm will be used to optimally assign the unit to the process based on force type, unit size and LAN assignment constraints. The mission data which consists of a set of activities, time durations and waypoint locations are utilized to drive the discrete event scheduler controlling the SAT simulation. The SAT simulates each of the missions for each of the units defined. Currently the SAT will not simulate resupply units but includes automatic resupply when the units supplies are depleted and a saved message to the user will occur. After the scenario is executed the user

has the ability to look at the resulting data and refine or change the given scenario. The SIU can be enabled so that the user can visualize the scenario as it unfolds. The user will have the capability to stop the scenario at a given point to save and analyze the data to determine whether or not to proceed. The SAT allows the planner to quickly determine scenario viability while providing the necessary network data to the IAT for network and infrastructure analysis. The following paragraphs will detail some of the outlined functionality described above.

2.2 SAT Scenario Initialization / Creation

The planner can create, load and edit exercise scenarios using the CLCGF SIU. The SIU runs as a separate process connecting to the SAT using sockets. Transmission Control Protocol / Internet Protocol (TCP/IP) datagrams are used by the SAT to send information updates to the SIU. The SIU enables the user to create unit representations of any type of force and entity type. Once the unit representation is defined it is added to the SIU and Table of Organization and Equipment (TOE) database. The databases are utilized by the planner to laydown the appropriate units for the specific exercise. When the desired units of a given scenario are placed on the terrain database the planner can save the SIU representation. This allows the capability to reload a particular scenario and make adjustments if necessary. When the planner is satisfied with the unit laydown it is saved and reformatted into data files to be read by the SAT. The data files which include the exercise, missions, units, processes and TOE files are used to initialize the SAT simulation. The exercise data file is used to initialize the scenario parameters such as data files to read in, database representation and parameters used to set simulation variables. The unit data file defines each of the units used in the scenario and their initial position. These units can be of any echelon, size, role and force. The missions file specifies the units movement, attack or scan activity each with an attached duration. The duration is based on the vehicles speed and distance between waypoints. The discrete event scheduler uses the mission data to execute the simulation. The processes data file specifies each of the units attached to each process. This will be created using the rule based algorithm mentioned earlier. During the initialization process of the SAT execution the unit data is sent to the SIU to place the units on the simulated terrain. The icons and colors signify the units' representation and strength. The user can then execute the simulation with the IAT enabled or disabled. If the

IAT is not selected an output file is generated to save the SAT data so that the IAT can run at a later time.

2.3 SAT Simulation Execution

The user starts by defining an exercise using the SIU or using a predefined scenario, the SAT reads in the necessary data files to execute the simulation. A mission process is executed for each unit. The units' waypoint, activity, and time duration of activity initiate the SAT process. The SAT schedules events based upon the activity's type and duration. Each of the waypoint route legs are divided into scheduled subevents. A full update of each of the units updates its location, velocity and interaction with other units. The units are currently divided into two categories, direct fire and indirect fire. The SAT also uses a TOE database lookup to provide relevant data to each of the unit types. This data includes entity composition, sensor range, engagement range, weapon impact range, and supplies. The direct fire unit's sensor range is utilized to determine if any units are within range. This is used to generate network traffic as well as to determine if an engagement can occur. For all units detected within sensor range, an update is performed. If this unit is an opposing force and within engage range an engagement is scheduled. During an engagement attrition is applied to each unit uniformly using Lanchester equations. Attrition is applied to supplies and entity counts based upon programmable parameters of defensive resistance and offensive power of each of the engaged units. Logistical consumption for petroleum, oil and lubricants (POL) as well as ammunition is also defined by the planner. During this engage activity, fire and detonate traffic is produced for infrastructure analysis. For indirect fire units an attack activity is scheduled and the weapon impact range is used to determine if and engagement of units within range are scheduled. The relevant network data is generated based on the interactions. The SAT simulation for each of the represented units occurs until one of the following events occurs: the unit is destroyed, missions complete or the planner stops the simulation.

2.4 Direct / Indirect Fire, Radar Units

Currently the SAT simulates three types of units based on the mission assigned. This includes direct fire, indirect fire and radar scanning units. Each unit is comprised of a combination of entity types. This data is stored in the TOE database. The ground unit types are divided into tanks, trucks, infantry fighting vehicles, dismounted infantry, and artillery vehicles.

Air units consist of fixed wing aircraft, unmanned aerial vehicles and rotary winged aircraft. This will be expanded to include other types. The SIU provides tools to create any unit type at any echelon level. These tools will eventually be integrated with the SAT software. A unit defined as indirect fire behaves differently than a direct fire. Some of the units can take on different missions. For example a FWA can be on a bombing mission in which case it is defined to be an indirect fire classification. An FWA can also be defined as flying against a defined ground target in which case it becomes a direct fire class.

Radar scan units include AWACs, JSTARS and any other intelligence gathering type unit.

2.5 Unit Movement

A simple algorithm is implemented for a units movement. A user defined route is used by defining a set of waypoints. Interpolation is used to move the unit from waypoint to waypoint. The units initial position in the units data file and the subsequent mission waypoints are used to simulate the units movement. Movements are currently limited by supplies, combat intensity and live/dead status. Attrition is applied to fuel consumption of a unit and is automatically resupplied when reaching a predefined resupply point. When a unit engages in combat with an opposing unit an attrition value is used to decrease the amount of supplies and strength. Once the unit has reached a user defined attrition percentage the unit stops its movement and a message is generated for the planner. Eventually the SAT will be integrated with a terrain module to use terrain factors to influence the units route planning and movement.

2.6 Unit Engagement

An engagement process links two units in the case of direct fire or a unit and an indirect fire position. An engagement occurs if the unit is within engage range and has strength and supplies. The process is initiated from full update. The duration of an engagement is estimated and a faster update rate is set. The duration is estimated as the minimum of the time for either unit to be destroyed, pass out of engagement range or the completion of an indirect fire activity. Engagement range is specified in the TOE data file for each type of unit. A weapon impact point range is used to simulate indirect fire units. The engagement will update the strength and logistical status of each unit (ammo and fuel consumed and destroyed). Attrition is modeled at very low fidelity, but follows the

standard service modeling practices. The losses are equal percentage losses based on relative fire power and time since last update. The next engagement is scheduled at the end of the interval. If a one sided engagement occurs, it continues as in the case of FWA attacking an undefended column of logistical trucks.

2.7 SAT Network Flow Data

The SAT produces network flow data based on the unit's activity and its interaction with relevant units. The data is in the form of process source, data type, flow rate, and process destination. The flow rate is based on Engineering Demonstration #1 (ED1) network traffic results. Currently the SAT generates the significant types of Distributed Interactive Simulation (DIS) Protocol Data Units (PDU). These include entity state, emission, transmitter, environmental, data collection, signal, fire and detonate PDUs. This data was used to validate the existing software.

2.8 SAT Analysis Data

As the initialized exercise progresses, SAT information is collected to provide information to the planner. The SAT can measure combat activity, losses, resupply points for each of the units participating in the exercise. The SAT saves scoreboard data which displays attrition for each of the units by alignment and its entities. The SAT produces diagnostic data if enabled for each of the runs. This data is in Excel spreadsheet format and tools will be developed to extract, reformat and analyze the data in order to be utilized efficiently by the planner.

2.9 SAT Visualization

The SAT utilizes the simulation created by the CLCGF effort. The SIU is used by the planner for exercise planning and initialization and also for the plan view display (PVD). Once a scenario is constructed the data is sent by the SIU to the SAT. The SAT simulation is started and the scenario is executed. As the SAT is executing the units' data is sent back to the SIU for display. The units are placed onto the SE database and tagged with the appropriate information (call sign, speed, location). As the move activities are executed for each of the units the position, velocity and strength of the unit is updated to the SIU PVD. This allows the planner to visualize the exercise as it executes. Once an engagement

occurs the strength is also updated. Color codes are used to signify the percentage of strength remaining for a unit. The unit turns black when all of its entities have been destroyed. The planner can use this data to determine the integrity of the scenario and replan if necessary.

The SAT also provides the user with a graphical user interface that provides status of the units including position, velocity, alignment, simulation process assigned, number of entities and supplies. This data is dynamically updated as the scenario progresses.

3. Infrastructure Analysis Tool

3.1 Functionality

The IAT has been designed to execute in conjunction with the SAT or in standalone mode, accepting as input the generated output of a previous SAT execution. The IAT accepts packet flow data and applies it to the network infrastructure. In order to model the packet flow data, the IAT requires a network topology and a list of simulation processes. Regardless of how the IAT is invoked, each IAT invocation is associated with a specific SAT scenario, consisting of one or more unit/mission pairs which are associated with one or more simulation processes.

3.1.1 Network Representation

The network infrastructure is specified by the user. Currently, this is done using data files. The network infrastructure consists of one or more LANs connected over a WAN. The two WAN configurations supported by the IAT are the Defense Simulation Internet (DSI) and the Advance Technology Demonstration (ATD) network / ACTS ATM network (AAI). Once a WAN is selected, the user can create sites with one or more LANs and connect each site to the WAN. The link connection types, such as T1, T3, etc., are configurable by the user. The three LAN configurations currently supported by the IAT are Ethernet, Switched Ethernet, and Fiber Distributed Data Interface (FDDI). For each LAN, the user specifies the number of nodes and the type of each node. The nodes can be routers or simulation hosts, such as Sun, Silicon Graphics (SGI), Hewlett-Packard (HP), and Digital (DEC) workstations or Network Personal Computers (PC).

3.1.2 Network Initialization

The IAT accepts the user-specified network topology and creates an aggregate network, which consists of nodes and links. The characteristics of the nodes and

links are initialized via a table lookup, based upon type. This information, which currently consists of commercially available data, is critical for the proper determination of processor and network loading. For nodes, this data includes the number of processors per node, the processor type, and the ratings per processor. For links, this data is the bandwidth of the link.

3.1.3 Assignment of Processes to Hosts

Given the network topology and the list of simulation processes, the user specifies the assignment of processes to simulation hosts' processors. The IAT provides the capability to save the current process/host assignments in order to be recalled for future executions of the selected SAT scenario.

3.1.4 Network and Host Loading

As the IAT executes, it computes the loading across the entire aggregate network of simulation nodes and links. The SAT provides the IAT packet flow rates from source to destination simulation processes. There are two types of packet flows: those which are added across the network, known as "Insertion" flows, and those which are removed from the network, known as "Deletion" flows. Using the user-specified process/host assignments (section 3.1.3), the IAT determines the "best" path from the source host to the destination host. The "best" path algorithm currently is based upon the fewest number of hops, but can easily be modified to include other variables such as current network traffic and cost. The IAT then applies the "Insertion" or "Deletion" packet flow to all nodes and links along the resultant path.

The network link calculation is simply specified as the data rate (bits per second). The simulation host loading calculation is based upon the number of entities simulated on the host and the remote packets received/processed by the host. The host loading equation was provided by MIT Lincoln Laboratories, based upon the results of ED #1.

3.1 Visualization

During the execution of a SAT scenario, the IAT displays the scenario's effect upon the specified network topology. The IAT provides a GUI which shows the network and host loading. For simulation hosts, the GUI displays their type, the assigned simulation process, and CPU loading. For links, the GUI displays the hosts connected by the link, their type, the bandwidth, and the current data rate in bits per second (bps). As the IAT executes and calculates

the CPU and network loading, it updates the GUI with the new load values. To signal to the user potential problems, the background color of the load entries changes between green, yellow, and red. These colors respectively represent less than 50% capacity, greater than 50% but not full capacity, and full capacity or overload.

3.1 Analysis

In the end, it is the IAT analysis data which is most important to the user. While the IAT provides a visualization capability which gives the user an idea of the network effects, it is the statistics it gathers and outputs which are crucial. Statistics currently gathered by the IAT include average flow rate between simulation hosts, average flow rate between all nodes, and peak flow rate between simulation hosts. Additional statistics to be gathered include the number of remote packets received per host, the number of packets received per LAN, and the number of packets transmitted over the WAN as a function of time. The analysis output can easily be input to an Excel spreadsheet. It should be noted that this type of analysis and output is time-consuming as it is routinely done by hand.

Another important statistic which the IAT will gather is simulation host processor loads over time. The IAT will use this to predict when simulation processes, such as ModSAF, are most likely to migrate entities to other simulation processes in order to balance the CPU load throughout the network.

3.2 Optimal Process/Host Assignment

The selected assignments of simulation processes to hosts can be a key determinant as to whether an exercise plan is or is not feasible. Different assignments can stress the infrastructure in different ways, and the user may wish to re-run the same SAT scenario with different process/host assignments so as to examine the type and level of stresses on the infrastructure. Since the user of the IAT is attempting to support large distributed exercises, exploring the alternative assignments of processes to hosts can be a laborious task.

In order to ease this process, the IAT will provide a utility which will suggest good process-to-host assignment for a given SAT scenario and network topology, for various different measures of stress on the infrastructure. The current design develops the initial assignment based on grouping units of similar

echelon and service that are close to each other in the initial scenario laydown. This is based on the two heuristics that the units are likely to share simulation support software and are likely to be close to each other in the simulated command hierarchy. sharing support software, or a common controller for several simulations makes it desirable that they be co-located for cost reasons. If they are closely linked in the command structure, it is likely that the units will stay close throughout the exercise. This means that they would be kept on the LAN to minimize traffic across the WAN.

After the SAT has been executed and a matrix of process-to-process flows over time is available, this assignment can be improved.

For example, individual processes can be swapped until no further reduction in stress is found. As the SAT provides data over time, it is now possible to modify assignments so as to reduce the peak loads on different LANs, even though they occur at different times, without having to consider only the average flows, or only the peak flows. In the first case, peaks would be ignored. In the second case, the user is forced to work under the false assumption that all inter-process flows peak simultaneously. We plan to include real-world constraints on swaps, such as required matching of specific process to specialized hosts, keeping simulators and simulations near their controllers, etc.

4. Planned Capabilities

The SAT/IAT is currently in BETA release. The following items outlines some of the major planned capabilities.

- The SAT will have the capability to test and analyze various multicast schemes developed under the STOW contract.
- SAT/IAT will have the capability to simulate the effects of aggregation/disaggregation and its impact on the network.
- The SAT/IAT will simulate the High Level Architecture (HLA) including insertion of the Federation object model for Interest management and support of data object passing.
- The SAT will be capable of simulating the effects of the Run-Time-Infrastructure (RTI) as well as the Simulation Support Framework (SSF) used for the STOW contract.
- Replace the IAT network topology data files with a GUI which provides the user with the ability to create, modify, and save network configurations. It will provide the user with the choices of WAN and LAN types as well as node types (routers, workstations, etc.).
- Future IAT visualization GUI enhancements will replace the network and host entry list with a connected network of nodes and links. The size of the nodes and links will increase or decrease based upon its corresponding load and data rate values. In addition, the colors of the nodes and links will change similar to the way the GUI currently does this for load entries.
- Enhance capability of SAT/IAT simulation based on STOW contract scheduled software release / task schedule.

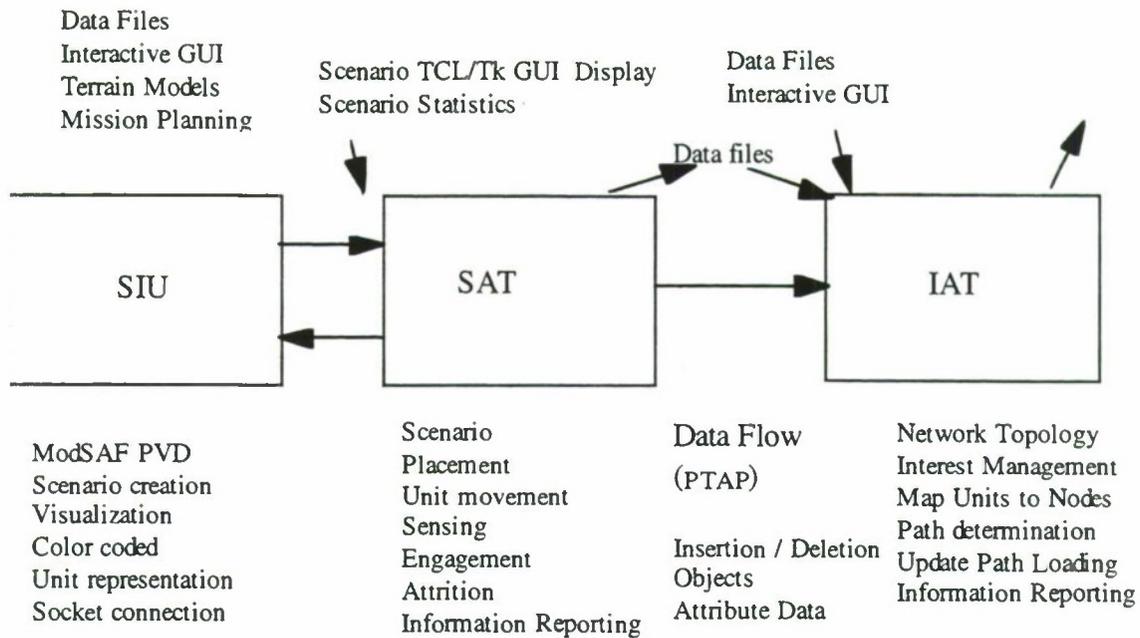


Figure 1 : SAT/IAT/SIU High Level Diagram

5. SAT/IAT Functionality

The illustration in Figure 1 shows the partitioning of the high level functions of the SAT, the IAT, and the CLCGF SIU as well as the data flow among these components. As mentioned earlier, the SAT and the SIU communicate using datagrams to create, initialize, and update a scenario. The SAT communicates with the IAT using packet flow rates, which are called PTAPs (source Process, data Type, flow Amount, and destination Process). This diagram also indicates the use of data files between the SAT and the IAT to allow either of these tools to run standalone.

6. Other related efforts

The SAT simulation is being utilized to support the JPSD DIS PDU to HLA translation effort. The SAT has been integrated with the translator and it will be used for testing the software by creating different scenarios to drive the Common software and the RTI.

7. Acknowledgment

This work is being done as part of ARPA's Synthetic Theater of War contract. STOW is intended to be the next great stride in Distributed Interactive Simulation.

STOW is focused on creating a large scale distributed simulation including detailed synthetic forces behaviors and detailed synthetic environment effects.

8. References

Peacock, Jeffrey Jr., Bombardier Kevin C., Panagos Jim, and Johnson Tom, (1996) "The JPSD Corps Level Computer Generated Forces CLCGF System Project Update", 6th Conference on Computer Generated Forces and Behavioral Representation.

9. Authors' Biographies

Robert D'Urso is a senior software engineer at Science Applications International Corporation's Technology Research Group (TRG) in Burlington, MA. He has been a key software developer for the SAT/IAT, as well as its predecessor SimTool, an SAIC IRAD that provided the basis for the SAT/IAT. He has also worked on the Fixed-Wing Aircraft behavior modeling for ModSAF 2.0. Prior to joining SAIC, Robert worked at Raytheon developing real-time embedded software for the PATRIOT Ground and Missile software. He is a graduate of Merrimack College in North Andover, MA., with a B.S. degree

in Computer Science. He also holds an M.S. degree in Computer Science from Boston University.

Michael Juliano is a senior software engineer in Science Applications International Corporation Technology Research Group (TRG) Burlington MA. Michael is the software lead for the SAT/IAT software development effort under the ARPA Synthetic Theater of War Program. He has been appointed as lead since joining SAIC in May of 1995. He is responsible for the management and software development of the SAT/IAT software. Prior to joining SAIC he has worked at Raytheon and Norden Systems developing and managing software through all phases on various radar and IR based signal processing real-time embedded software systems. He is a graduate from the University of New Haven, West Haven, CT. with a B.S. in Computer Science and is currently attending Boston University towards a Master s Degree in Computer Science.

Dr. Edward Powell received his Bachelors of Science degree from Carnegie-Mellon University and his Ph.D. in Astrophysics from Princeton University. He has previously worked at Lawrence Livermore National Laboratory's Conflict Simulation Laboratory where he was involved in analytic and distributed simulation research using the Joint Conflict Model. He is currently a senior scientist with Science Applications International Corporation, where he led the Joint Precision Strike Demonstration Run-Time Infrastructure effort and is now a lead architect on the Synthetic Theater of War Program.

Dr. Ben Wise is a Senior Scientist at SAIC's TRG group in Burlington MA. He received a B.S. in Physics from MIT and a Ph.D. in Engineering and Public Policy from CMU. He taught graduate operations research, probability and statistics, and artificial intelligence at Dartmouth College before entering the commercial sector. He worked on corps-level battle simulation and strike planning tools while at McDonnell Douglas, before moving to BBN and assisting the SimNet, Odin, ModSAF, and Warbreaker projects. He moved to SAIC in 1993 and started a new office specializing in advanced simulation and planning technologies. He designed the Patriot and scud simulations for BBN's Warbreaker effort, the fixed and rotary wing aircraft dynamics for LORAL's ModSAF, prototyped the tactics representation language for Warbreaker. He served SAIC as the initial lead in the Corps Level Computer Generated Forces, Command Forces, and SAT/IAT projects. He is currently involved in several projects focusing on issues in the linkage of

constructive and virtual simulations. His research interests focus on planning under competition and uncertainty.

QuickSet: A Multimodal Interface for Military Simulation

James A. Pittman, Ira Smith, Phil Cohen, Sharon Oviatt, Tzu-Chieh Yang
Center for Human Computer Communication
Oregon Graduate Institute
P.O.Box 91000, Portland, OR 97291-1000
jay@cse.ogi.edu

1. Abstract

We are developing a hand-held system to control ModSAF and its 3D visualization. The interface, called QuickSet, combines speech and pen-based gesture input, and allows the user to set up training exercises by creating forces and control measures, and to control the exercise by assigning tasks to the forces. The interface consists of a PDA (a hand-held PC weighing roughly 3 lbs) employing wireless LAN communications, color screen, microphone, pen stylus, on-board speech recognition, and on-board gesture recognition. Communication between the military simulation system and the PDA is brokered by the Open Agent Architecture.

The design of speech commands and the pen gestures is driven by observations of real users using a simulated interface. These observations occur in Wizard-of-Oz experiments in which a human collaborator plays the role of the speech and gesture recognizers. This methodology allows us to tailor the interface to the users, and to do so before the system is built. Subsequently, the system is built and tried by real users.

2. Introduction

The STOW-97 exercise anticipates substantial expansion in the number and types of entities to be created and simulated. However, the graphical user interface paradigm employed in the core simulator (ModSAF (Courtemanche and Ceranowicz, 1995)) will not scale easily to larger exercises. For example, the following (Figure 1) is the entity creation *menu* for MCSAF (USMC ModSAF). Note that for company-level training with LeatherNet the menu fills a 21" SGI screen.

Clearly the ModSAF GUI as currently formulated has exceeded the design parameters for effective GUI interaction. Given that users also express a desire for a small *portable* device for simulation set-up, a new user interface paradigm needs to be employed. We propose use of multimodal interaction--employing

speech, pen, and GUI technologies as best fits the problem at hand.



Figure 1: One menu from the MCSAF unit editor.

Our goal is to build a hand-held system that can be used to control LeatherNet, the USMC training simulation facility developed at NRAd (Clarkson, 1996), which employs ModSAF for combat simulation and CommandVu, an NRAd-enhanced version of NPSNET (Zyda, Pratt, Monahan, and Wilson, 1992), for 3D terrain visualization. Recently SRI International added the CommandTalk system (Moore, 1995), which provides speech input based on Marine Corps radio communication.

CommandVu is controlled by a special GUI called VABS. VABS attempts to make available the extensive functionality of CommandVu with a small numerical keyboard, and thus users must rely on a labyrinth of modes and mode-switching keys. In addition, some CommandVu commands require the selection of a vehicle by mouse click. Fast moving vehicles have proven difficult to select.

QuickSet combines speech and pen-based gesture input, and allows the user to set up LeatherNet training exercises by creating forces and control measures, and to control the exercise by assigning tasks to the forces (Cohen, Pittman, Smith, and

Yang, 1996). It also allows the user to control the viewpoint of the CommandVu terrain visualization, along with its many features (radar, HUD, tethered modes, etc.), and also provides control of video switching for large-screen displays. The system consists of a PDA (a hand-held PC weighing roughly 3 lbs) employing wireless LAN communications, color screen, microphone, pen stylus, on-board speech recognition, and on-board gesture recognition. Communication between ModSAF and the PDA is brokered by the Open Agent Architecture (Cohen, Cheyer, Wang, and Baeg, 1994). Figure 2 shows an artist's rendition of QuickSet controlling the LeatherNet "Cave" display, and Figure 3 shows collaborative use of the system for creating units, areas, lines, and a minefield breach.

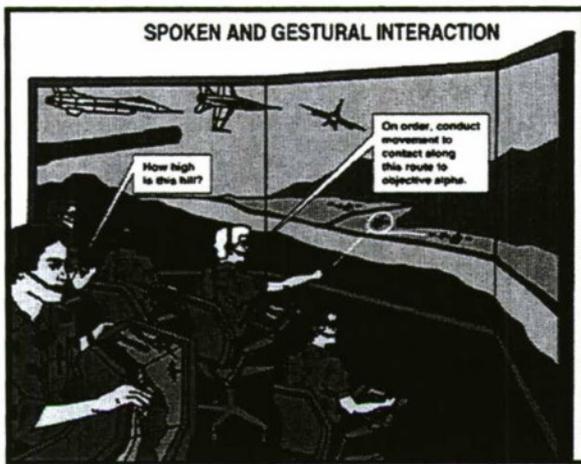


Figure 2: Artist's rendition of multimodal interaction with 3D simulation, 2D maps on PDAs, and a "deployable unit."



Figure 3: Using QuickSet for multimodal, collaborative simulation set-up.

Speech input and pen input each have advantages and disadvantages. Speech enables naming things not

currently visible on the screen, such as platoons just out of view on a map, or tasks, procedures, rules, or situations that do not have an iconic presentation. Speech is also faster for issuing commands. Pen input is often more convenient (and more accurate) for indicating objects that are currently in view on the screen. Pen input also allows specification of irregular lines that might indicate routes, or boundaries of areas such as minefields, swamps, landing zones, assembly areas, etc. Moreover pen input has the advantage of being usable in public places, where one might not want to verbalize commands because of privacy or secrecy, and also is usable where the noise of weapons, aircraft, and ground vehicles can prevent use of a speech recognizer.

Our multimodal input system gives the user the ability to capitalize on both sets of advantages, using whichever modality meets the need of the moment. In addition, experimental subjects often switch modes to deal with error correction, writing a word that the speech system just cannot recognize, or uttering a word that the pen system fails to recognize.

3. High-Fidelity User Interface Simulations

We have investigated the use of speech-only, pen-only, and combined speech-pen input modalities in a variety of tasks, using high-fidelity Wizard-of-Oz simulations (Oviatt, 1996; Oviatt, Cohen, Fong, and Frank, 1992; Oviatt, Cohen, and Wang, 1994). In these experiments, subjects perform tasks such as making airline reservations, checking bank accounts, or updating maps and locating houses on maps for real estate clients. The subjects use what they believe to be a speech recognizer and a pen gesture recognizer to enter commands. In fact, the recognition is performed by a collaborator hiding in an adjacent room.

These studies demonstrate that combined speech-pen interfaces, as compared to speech-only or pen-only interfaces, reduce user's wordiness, utterance length, lexical variability, disfluencies, bigram perplexity, and syntactic ambiguity (i.e., number of parses generated). This yields significantly faster task performance, and significantly fewer user errors. Not surprisingly, experiment subjects show a strong preference for the multimodal interface, as can be seen in Figure 4.

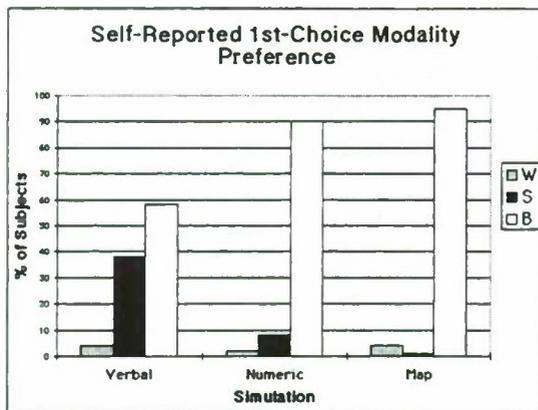


Figure 4: Self-reported preference for multimodal versus unimodal interaction in verbal, quantitative, and map-based simulations.

4. The Open Agent Architecture

The Open Agent Architecture (OAA) (Cohen, Cheyer, Wang, and Baeg, 1994) is based on FLiPSiDE (Schwartz, 1993), an enhanced blackboard architecture. In the traditional blackboard model individual knowledge sources (agents) communicate by posting and reading messages on a common blackboard. An agent will periodically poll the board to see if there are any posted goals (from other agents) it can solve; when an agent needs help, it can post a goal to be solved, then retrieve the answer when it appears on the board. The OAA model enhances this with a facilitator agent resident on the blackboard. This facilitator stores the blackboard data, identifies agents that can solve particular posted goals and routes requests to the appropriate agents.

In the Open Agent architecture all communication among the agents takes place through the blackboard and its facilitator agent. In addition to the standard blackboard operations of posting and reading, agents in an OAA can send general and specific queries to the blackboard's facilitator agent and they can have the facilitator set triggers. A general query asks the facilitator agent to route the goal to be solved to any or all agents that can solve it, a specific query tells the facilitator to route the query to a specified agent, and by setting a trigger an agent is asking the facilitator to notify it when a specific event has occurred. The OAA uses an interagent communication language (ICL) that consists of horn clauses. The language is a standard prolog enhanced with certain temporal operators.

Under the Open Agent Architecture, when an agent joins a blackboard, it registers with the blackboard, and with the facilitator agent, by providing a list of goals it can solve, and (optionally) with a list of goals to which the agent wishes to subscribe; the facilitator will add the agent to its list of available knowledge sources. Whenever a goal to be solved is posted to the blackboard, the facilitator routes the goal to a subset of those registered agents that have claimed to be able to solve it. When a message is posted to the blackboard by an agent, the facilitator will route the message to all the agents that have subscribed to messages of that type. The OAA's facilitated architecture allows blackboard communication to be more efficient than in a standard blackboard architecture--agents no longer have to continually poll the board, their help will be requested when a goal for them to solve is posted, and they will be notified when messages for them, either requested solutions or predicates they have subscribed to, are posted to the blackboard. Agents only need to initiate communication with the blackboard when they have a request to make of another agent, or when they need a predicate they do not subscribe to.

The Open Agent Architecture is a flexible system that provides a means for "agentifying" previously written programs through a library containing the basic features of the OAA's ICL. This library can be linked with existing programs, allowing a legacy program to function as an OAA agent. Libraries currently exist for programs written in Prolog, C, C++, Visual Basic, and Java.

In its QuickSet implementation, the OAA uses ten primary agents to control the simulation and to provide windows into the simulation via the world wide web and CommandVu. The QuickSet agent configuration is illustrated in Figure 5. The natural language agent (Gemini (Dowding et. al. 1993)), the multimodal interpretation agent, an agent to execute the logical forms produced by the natural language agent, the agentified ModSAF, and the PDA-based agents (the user interface agent (UI agent), gesture recognizer and the speech recognizer) are used to control the ModSAF simulation. The other agents (video controller, web agent, and CommandVu agent) are used to control the various user interfaces that are displaying the simulation. The blackboard, Gemini, the logical form agent, and the agentified version of ModSAF are all provided by SRI International.

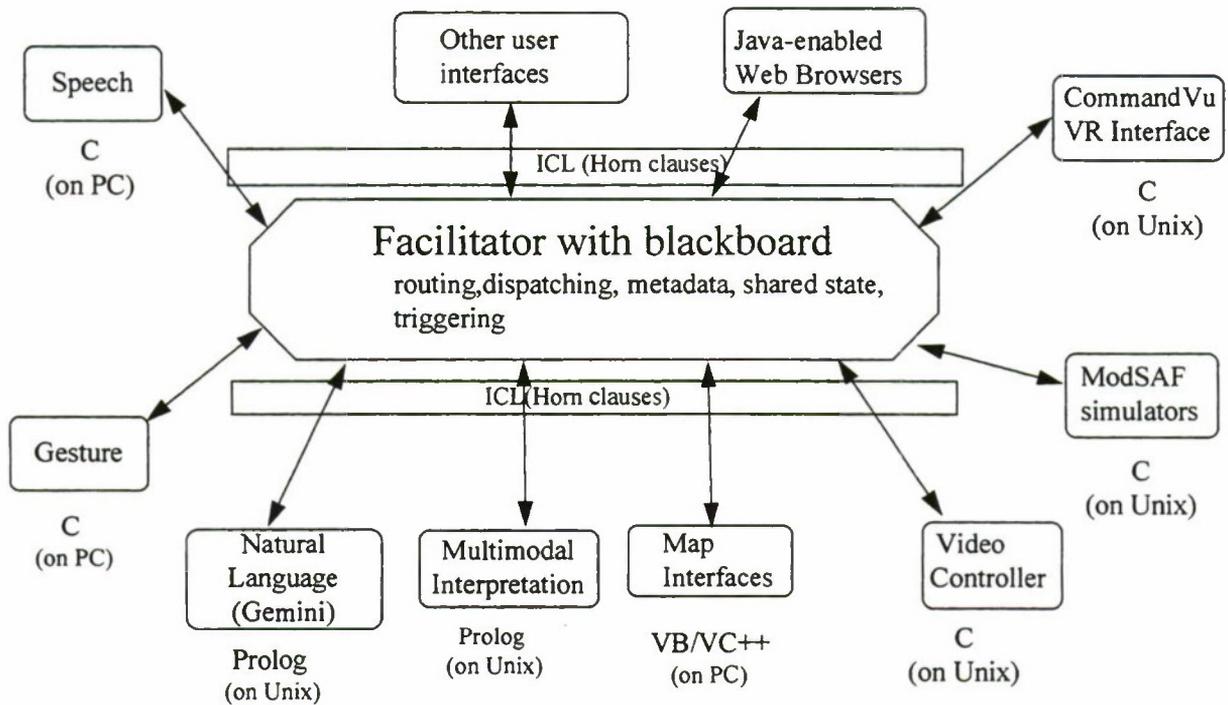


Figure 5: The blackboard serves as a facilitator, channeling queries to agents who claim they can solve them.

The QuickSet system is run in a multi-platform, multi-os environment; the PDA agents are in a Windows95 environment, ModSAF and CommandVu run on SGI platforms under IRIX, and the other agents can run in either the SGI environment or on SparcStations under SunOS.

An agent's ability to subscribe to one or more predicates provides the basis for an initial collaboration facility among users. When a user enters the QuickSet system, the PDA's UI agent subscribes to the ink predicate. Whenever any PDA in the system produces ink, the blackboard's facilitator agent will route the ink to every other PDA. In this way, all user input appears on every PDA; every user is made aware of the totality of input in the system.

4.1 World Wide Web Access to ModSAF

The web agent allows an interested observer to watch the simulation from any workstation supporting a Java-enabled World Wide Web browser, as seen in

Figure 6. The web agent is an agentified Java applet embedded in a web page. Because NetScape constrains Java applets to interact only with their home domain, a web server is run on the same host as the QuickSet blackboard. Communicating through the server, the web agent queries the blackboard for information (e.g. unit positions, objectives, lines of departure). The blackboard routes the request to the ModSAF agent, then routes the answer back to the web agent. By repeatedly querying for current information, the web agent is able to maintain an up-to-date display of the status of the simulation. Our future plans include enhancing this agent to enable it to update the simulation through pen/voice interaction.

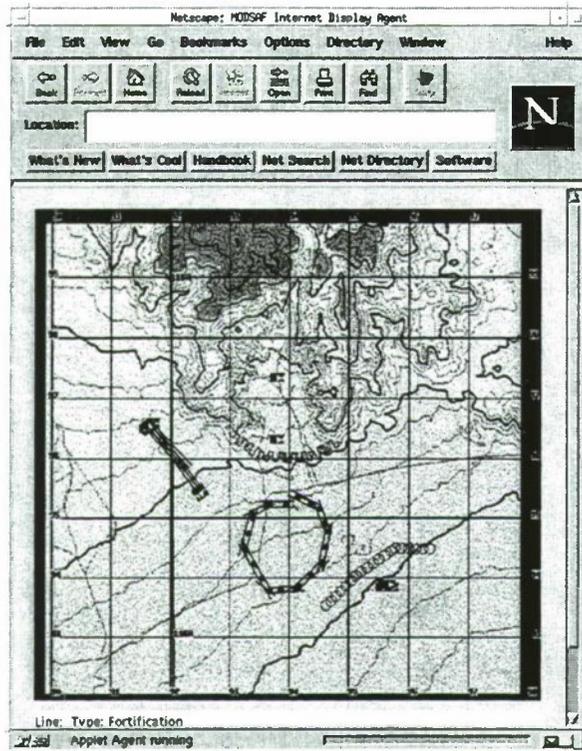


Figure 6: The Java applet shows the ModSAF map, units, lines, and objectives.

4.2 Using the QuickSet PDA

Under the QuickSet system, the ModSAF simulation is controlled using the PDA. After the user interface agent (UI agent) has connected to the blackboard, an image of the current exercise's terrain appears on the PDA screen (Figure 6). The PDA user can control the ModSAF simulation by a combination of gesture and speech directed at the PDA. For instance, to create a new unit a user might click the pen on the map (at the spot the unit is to be placed) and utter "m one a one platoon". The UI agent directs the speech stream to the speech agent for conversion to a list of recognized words; at the same time, the pen click is directed to the gesture agent. The information in the two streams are combined (the integration process is described in detail in the sequel), and the resultant command is sent to the ModSAF simulation.

In addition to controlling the simulation, a PDA user can effect other simulation visualization tools. Speech commands issued through the PDA (e.g. "commandvu world view on", "commandvu go to one thousand") are routed to the CommandVu agent, controlling the CommandVu display. A video switching agent controls the video signal reaching a large display monitor in our demonstration room.

Currently the video switching agent can choose between the ModSAF display, the web agent, CommandVu, and one PDA screen.

5. The QuickSet Gesture Recognizer

The QuickSet gesture recognizer consists of a neural network and a set of hidden Markov models. For the neural network recognizer the gesture is size normalized, centered in a 2D image, and fed into the neural network as pixels (Pittman, 1991). For the HMM recognizer the ink is smoothed, resampled, and converted to deltas, and fed to the HMM recognizer.

Both recognizers provide the same coverage (they recognize the same set of gestures). These gestures, some of which are illustrated in Figure 7, include various military map symbols (platoon, mortar, fortified line, etc.), editing gestures (deletion, grouping), route indications, area indications, taps, lassos, etc. The probability estimates from the two recognizers are combined to yield probabilities for each of the possible interpretations.

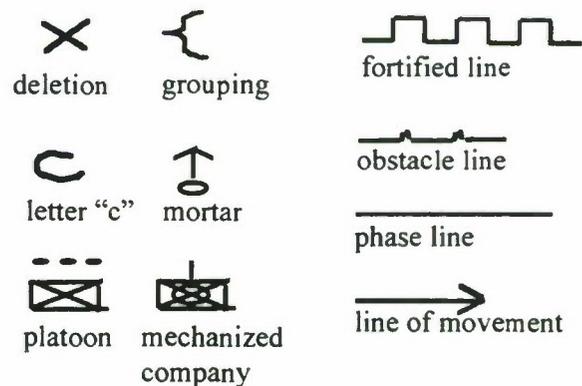


Figure 7: Some of the symbols and gestures used in QuickSet.

The inclusion of route and area indications creates a special problem for the recognizers. Both recognizers recognize shape (although they see the shape in different data formats). But as Figure 8 shows, route and area indications may have a variety of shapes. This problem is further compounded by the fact that we want the recognizer to be robust in the face of sloppy writing. More typical, sloppy forms of various map symbols, such as are illustrated in Figure 9, will often take the same shape as some route and area indications. A solution for this problem can be found by combining the outputs from the gesture recognizer with the outputs from the

speech recognizer, as is described in the following section.

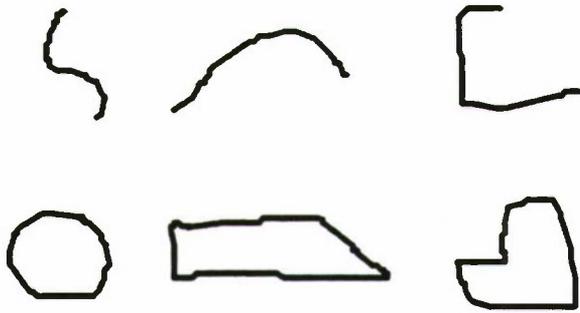


Figure 8: Pen drawings of routes and areas. Routes and areas do not have signature shapes that can be used to identify them.

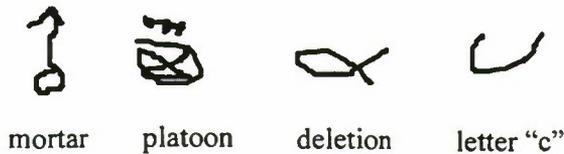


Figure 9: Typical pen input from real users. The recognizer must be robust in the face of sloppy input.

6. Integrating Speech And Gesture Recognizers

Multimodal input allows error correction by users by switching modes. But it also enables another type of error correction, an automatic form of error correction in which the system combines roughly concurrent pen gestures and speech utterances to form a single multimodal command.

For instance, the user might speak the command: "M1A1 platoon follow this route" while concurrently drawing a route with the pen from the platoon to some objective. This is illustrated in Figure 10. Depending on the shape of the route, the gesture recognizer might have (mis)recognized this gesture as a route, an area, a tap, a letter or digit, a map symbol, or an editing gesture. The interpretation with the highest probability is shown on the PDA. But as seen in Figure 11, the gesture recognizer also issues a logical form to the blackboard that indicates the probabilities of each of the interpretations. Those interpretations that involve objects on the screen (such as taps and group encirclings, called "lassos") include the object IDs of those objects. In the future we will add similar

logical forms to represent various interpretations of the speech recognizer, along with their probabilities.

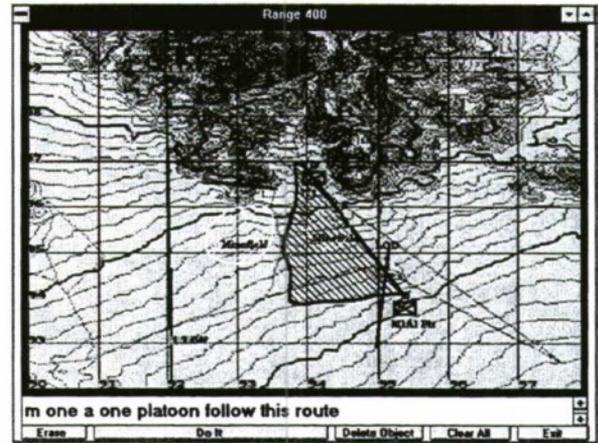


Figure 10: The user enters a multimodal command. The ink and its interpretation are shown in place on the map. In this example the ink was misrecognized as an area. The user's speech (as recognized) is displayed at the bottom. The user can correct speech misrecognitions with the pen.

```
gesture(area(0.65, [...coords...]),
objects(0.65, [...IDs...]),
line(0.50, [...coords....]),
grouping(0.45, [...IDs...]),
letter(0.25, 'c'),
unit(0.21, mortar),
letter(0.21, 'o'),
digit(0.21, 0),
deletion(0.11, ID),
tap(0.01, ID)).
```

Figure 11: Logical form issued to the blackboard by the gesture recognizer. Each interpretation has a probability. Note that the probabilities do not sum to 1.0, as they are not mutually exclusive interpretations from the viewpoint of the gesture recognizer.

7. Summary

Our multimodal interface has been implemented on a wireless hand-held PDA, and interfaced to the United States Marine Corps version of ModSAF and CommandVu, called LeatherNet, as well as the United States Army version of ModSAF. The interface supports the creation of units and control measures, the issuing of tasks, the maneuvering and control of CommandVu's stealth viewpoint, and the control of the video feed to a large display screen.

The Open Agent Architecture enables collaborative interactions among users. In addition, we have created a Java-enabled web page connected via the agent architecture that allows users to view the simulation via a web browser.

Our research continues, focusing on improving and integrating speech recognition, pen gesture recognition, and natural language understanding, and on improving and clarifying the use of agent architectures as a foundation for this integration.

The project will be demonstrating the value of the research by delivering a working system to the USMC training facility at 29 Palms, California for actual use in training. We have already demonstrated the system at the Royal Dragon exercise at Ft. Bragg. In addition we will deliver our multimodal interface to other DARPA-supported research projects involved in military training exercises. To support this we will continue to collect more gesture data from real users, to add gestures to the gesture vocabulary, and to add more commands to the speech system.

8. Acknowledgments

This work is supported by the Information Technology and Information Systems offices of DARPA under contract number DABT63-95-C-007, and has been done in collaboration with the US Navy's NCCOSC RDT&E Division (NRaD) and SRI International. Thanks are extended to Richard Wesson and Ed Stuber for implementing the Java-OAA connection.

9. References

Clarkson, J. (1996) LeatherNet: A synthetic forces tactical training system for the USMC commander. *Proceedings of the Sixth Conference on Computer Generated Forces and Behavioral Representation*. University of Central Florida.

Cohen, P.R., Cheyer, A., Wang, M., and Baeg, S.C. (1994) An Open Agent Architecture. *Proceedings of the AAAI Spring Symposium Series on Software*. Stanford University, CA, 1-8.

Cohen, P. R., Pittman, J. A., Smith, I., and Yang, T. C. (1996) Multimodal interaction for distributed interactive simulation. In submission.

Courtemanche, A. J. and Ceranowicz, A. (1995) ModSAF development status. *Proceedings of*

the Fifth Conference on Computer Generated Forces and Behavioral Representation. University of Central Florida, 3-13.

Dowding, J., Gawron, J. M., Applet, D., Bear, J., Cherny, L., Moore, R., Moran, D. (1993) Gemini: A natural language system for spoken language understanding. *Proceedings of the 31st Annual Meeting of the ACL*.

Moore, R. C. (1995) CommandTalk: Spoken language interface to the LeatherNet system. *ARPA Software Technology and Intelligent Systems Symposium*, Chantilly, Virginia, 28-31 August.

Oviatt, S.L. (1996) Multimodal interfaces for dynamic interactive maps. *Proceedings of CHI'96 Human Factors in Computing Systems*, ACM Press, NY, 95-102.

Oviatt, S. L., Cohen, P. R., Fong, M. W., and Frank, M. P. (1992) A rapid semi-automatic simulation technique for interactive speech and handwriting. *Proceedings of the 1992 International Conference on Spoken Language Processing, vol 2*. J. Ohala (ed.), 1351-1354.

Oviatt, S.L., Cohen, P.R., and Wang, M. (1994) Toward interface design for human language technology: Modality and structure as determinants of linguistic complexity. *Speech Communication* 15, 3-4, 283-300.

Pittman, J. A. (1991) Recognizing handwritten text. In *Proceedings of CHI'91 Human Factors in Computing Systems*, ACM/SIGCHI, NY, 271-275.

Schwartz, D. G. (1993) Cooperating heterogeneous systems: A blackboard-based meta approach. *Technical report 93-112*, Center for Automation and Intelligent Systems Research, Case Western Reserve University, Cleveland, Ohio, Ph.D. thesis.

Zyda, M. J., Pratt, D. R., Monahan, J. G., and Wilson, K. P. (1992) NPSNET: Constructing a 3-D virtual world. *Proceedings of the 1992 Symposium on Interactive 3-D Graphics*.

10. Authors' Biographies

James A. Pittman is a Senior Research Associate at the Oregon Graduate Institute. He has a Ph.D. in human-computer interface. His research interests include gesture recognition, character recognition, and pen-based interfaces.

Ira Smith is a Ph.D. candidate at the Oregon Graduate Institute. He has a Masters of Science degree in computer science. His research interests

include agent architectures, inter-agent communication languages, dynamic logics, distributed objects, and multimodal interfaces.

Phil Cohen is Professor and Director of the Center for Human-Computer Communication. He has a Ph.D. in computer science. His research interests include multimodal interfaces, human-computer interaction, intelligent agents, dialogue, natural language processing, collaboration theory and technology, speech act theory, delegation technology, knowledge-based simulation applications to mobile computing, information management, and manufacturing.

Sharon Oviatt is Associate Professor at the Oregon

Graduate Institute. She has a Ph.D. in experimental psychology. Her research interests include human language technology and multimodal systems, modality effects in communication (speech, writing, keyboards, etc.), communication models, telecommunications and technology-mediated communication, interactive systems, human-computer interaction, empirically-based design and evaluation of human-computer interfaces, cognitive science, and research methodology.

Tzu-Chieh Yang has just received his Masters degree in computer science from the Oregon Graduate Institute. His research interests include integration of speech and pen interfaces.

Soldier Station: Integrating Constructive and Virtual Models

Shirley Pratt and David Pratt
Department of Computer Science, Naval Postgraduate School, Monterey, CA 93943
pratts@cs.nps.navy.mil pratt@cs.nps.navy.mil

David Ohman and John Galloway
TRADOC Analysis Center, ATRC-WAC, White Sands Missile Range, NM 88002
ohman@trac.wsmr.army.mil gallowaj@trac.wsmr.army.mil

1. Abstract

Soldier Station is a unique simulation system which bridges the gap between two distinct realms of modeling: constructive and virtual. The Soldier Station operator controls a simulated dismounted infantry soldier in a 3D virtual environment with rules of movement, engagement and tactics provided from a constructive model. This paper describes the Soldier Station system, its design, and the integration of two separate simulations with radically different modeling philosophies. The features of the resulting system, its limitations, and plans for future work are presented.

2. Introduction

Traditional US Army simulations generally fall into one of two distinct modeling realms: constructive or virtual. Constructive simulations allow a user the ability to control one or more battlefield entities subject to software rules, data and procedures. Entities are indirectly controlled through the user's interactions with a 2D plan view display and a Graphical User Interface (GUI). In contrast, virtual simulations strive to immerse a user into a 3D synthetic environment as the entity itself. The user interacts with various input devices which provide direct control over the entity. Currently, systems of both types of simulations are in popular use. Major efforts have been expended to allow these disparate systems the ability to participate together in joint Distributed Interactive Simulation (DIS) exercises. Soldier Station is a unique effort which brings the two modeling realms together within a single DIS-compatible system.

Soldier Station bridges the gap between the two modeling realms by integrating together the US Army's constructive Janus model algorithms and the Naval Postgraduate School's NPSNET virtual envi-

ronment system. It allows for visual realism and user interactivity that is currently not available in standard US Army constructive models. It utilizes realistic movement, detection and engagement algorithms not present in most virtual simulators. The integration of two well established simulations represents a significant reduction in project risk while offering significant advantages over building either system independently. The primary purpose of Soldier Station is to serve as an analytic tool for TRADOC Analysis Center (TRAC) to address Land Warrior program issues concerning Dismounted Infantry (DI) command and control, situational awareness, tactics, techniques and procedures.

Soldier Station is DIS-compatible and able to interoperate with other constructive and virtual systems which use DIS Version 2.0.3 or 2.0.4 network protocols. The availability of a particular terrain database format required by an application, however, can be another limiting factor to this interoperability. As part of this project, a terrain tool was developed to convert the gridded Janus terrain data into a polygonal database format appropriate for many visual simulations including NPSNET and Soldier Station. Generally speaking, terrain format incompatibilities are a major problem in the DIS simulation community which exceeds the scope of this paper.

3. System Overview

Soldier Station is actually a system of systems. It is designed to run on two separate Silicon Graphics (SGI) workstations, a multiple processor SGI Onyx Reality Engine2 and a SGI Indy. Two machines are used to accommodate the large graphics and CPU processing requirements of the main simulation system and to meet substantial user interface system demands. One SGI Onyx with a multi-channel option (MCO) is actually more expensive and more likely to become overloaded resulting in poor system performance than a two machine system.

3.1 User Interface Components

Figure 1 shows the various user interface components of Soldier Station. The visual display is a 3D perspective view of the synthetic environment. This view depends on the DI entity's posture, head and body orientation and the current sensor. From two nearby speakers the user can hear DIS networked battlefield sounds. Verbal communication with other DIS participants is possible using a telephone or radio headset.

has three degrees of freedom allowing body and head/weapon orientation (heading and pitch). It also has a trigger for weapon firing capabilities. From the touch screen GUI the user may easily select different types of weapons, sensors or instruct the DI entity to execute one of numerous different types of hand signals. To orient himself on the battlefield, there is a 2D map which can show various information overlays and the relative position of other detected or dead entities. The GUI also has a compass which indicates the current body and head/weapon orientation, and various textual feedback information (i.e. location coordinates, actual speed of travel, ammunition rounds left, movement/injury status). Table 1 sum-

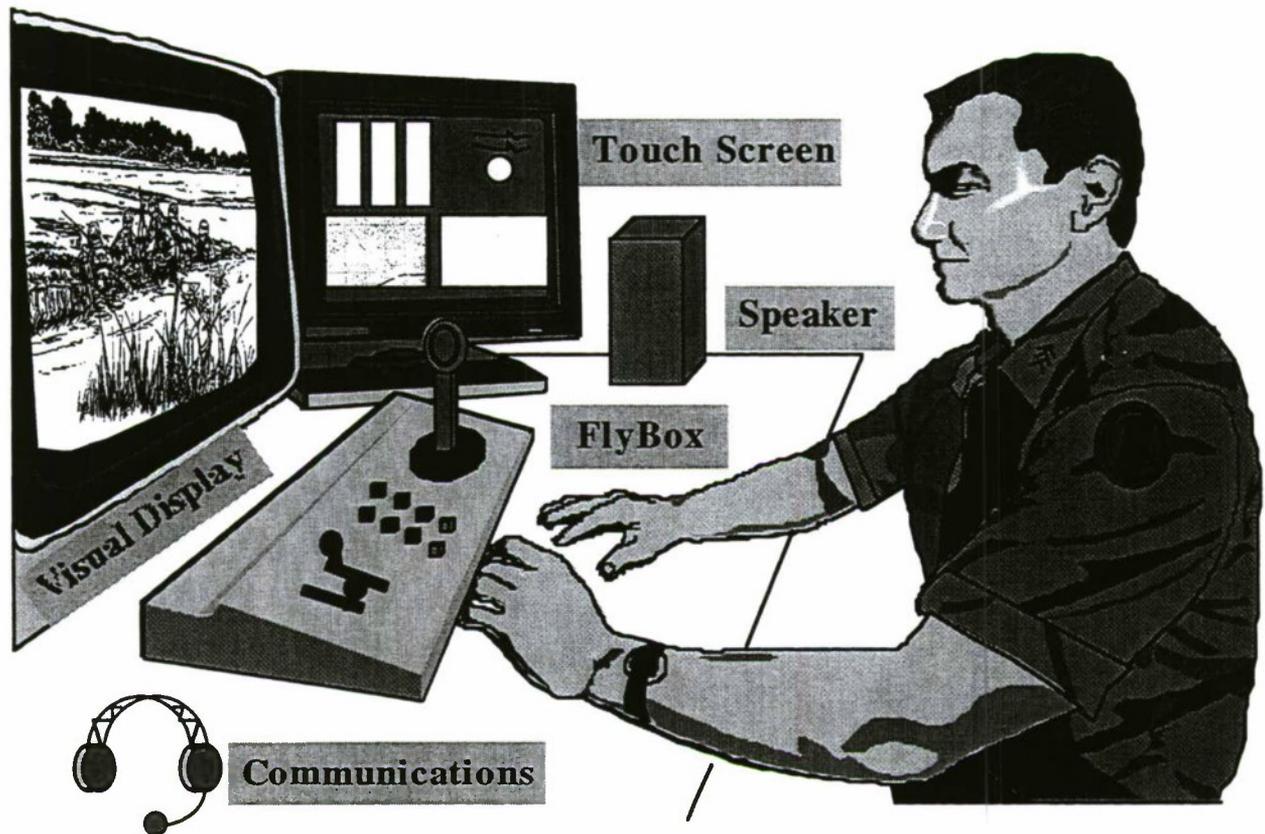


Figure 1: User Interface Components of Soldier Station

The operator controls the input speed and the soldier's posture using levers and switches on the BG Systems Flybox input device. The Flybox joystick

marizes some of the various options which the Soldier Station operator may select while controlling the DI.

3.2 Software Components

The software components of Soldier Station are shown in Figure 2. The main system runs on the SGI Onyx and is comprised of two tightly coupled modules, a Visualization Module (VM) and a Combat Module (CM). The User Interface System (UIS) runs on the SGI Indy and consists of two separate

the FORTRAN algorithms for Soldier Station system initialization, DI movement, detection, weapon firing and damage assessment. These routines, which were originally part of Verified and Validated (V&V) Janus Version 4.2 code, were modified to allow enhanced user control over the DI entity. Later they were upgraded to Janus Version 6.0 which most notably supports multiple sides of forces and fratricide. Although the modified routines are still subject to the

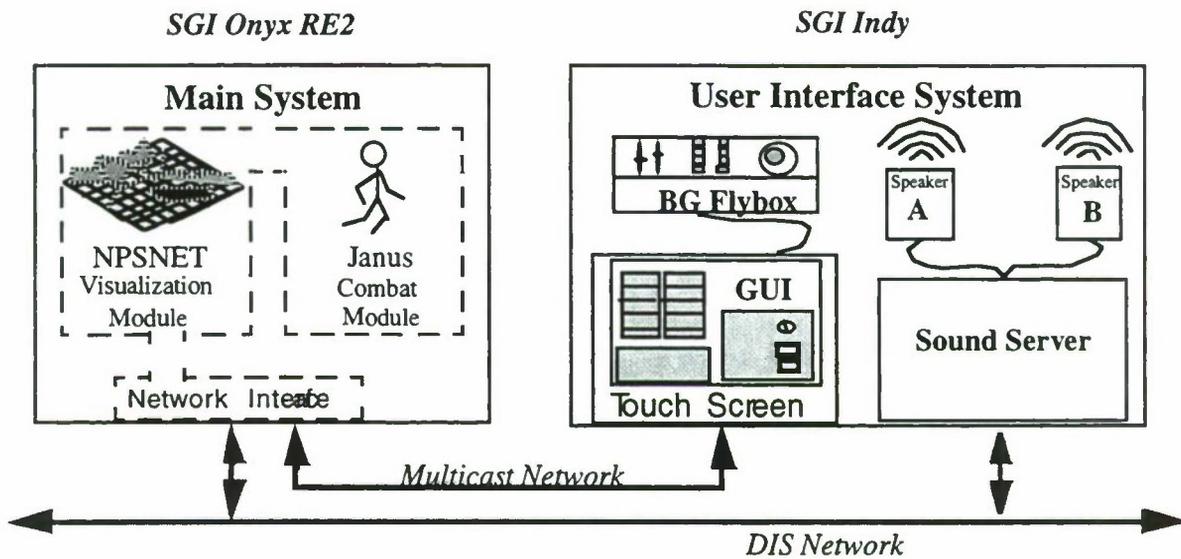


Figure 2: Software Components of Soldier Station

applications, a GUI and a sound server.

The VM is responsible for overall Soldier Station program control. It is a modified version of Naval Postgraduate School's NPSNET Version IV.8 system (Pratt et al., 1996b). NPSNET is an object oriented C++ application which uses the SGI Performer visual simulation toolkit (Rohlf and Helman, 1994) to create 3D graphical representations of terrain, objects, entities and environmental effects. The VM also provides DIS network management, remote entity dead reckoning (DR) and simulation of the local DI entity. Soldiers are generally represented using a medium resolution, fully-articulated soldier model and a library of real-time animations from University of Pennsylvania's Jack system (Granieri and Badler, 1995).

The Combat Module (CM) is based on the US Army's Janus system (US Army, 1996). It contains

V&V process, they provide realistic feedbacks for maneuvering over terrain and obstacles, for weapons firing outcomes and injury determinations which were not previously present in NPSNET.

The GUI application acts a central collection point for all of the user input made via the BG Flybox and touch screen devices. It packages the inputs into Interface Data Units (IDU) protocols and sends them to the main system via multicasting. The VM processes the inputs, makes the appropriate calls to the CM routines and then sends back feedback data in IDUs for the GUI to display. Thus, it is possible for the operator to request unrealistic speeds, postures or weapon firing given certain situations and be limited by the CM which allows, in theory, only reasonable outcomes to occur.

In addition to the feedback data displayed on the GUI, the user hears various battlefield sounds from an

NPSNET sound server application running on the SGI Indy. The sound server listens for certain DIS PDUs which it recognizes as having a sound associated with them (primarily Fire and Detonation PDUs and also some of the local entity's Entity State PDUs). Upon receiving such PDUs, the distance between the DI entity and the source of the sound is computed as the sound wave propagates in the virtual environment. When the distance is zero, the sound is played over the speakers adding considerable realism to the simulation.

3.3 System Design Considerations

The Soldier Station system is designed to host the VM and CM together as a single integrated application running on the SGI Onyx. The modules interact extensively during the simulation and pass considerable amounts of information between them. In order to minimize the communication latency between the two modules, the CM routines are bundled into a library and linked to the VM.

Substantial user interface requirements for the Soldier Station system justify the use of a second low-end SGI workstation. A separate monitor is needed for a touch screen. The GUI application manages inputs from both the touch screen and the BG Flybox devices, passes user input information to the main system, and receives and displays feedback information from the main system. In addition, the sound server application requires a minimum of 32 MB memory to be able to play sounds instantaneously upon receiving the appropriate PDUs. These demands and the high cost of a new SGI MCO display drove the decision to run the UIS system on a separate SGI Indy workstation. This two machine configuration frees valuable computational resources for the main system which has substantial demands for graphics, entity simulation and networking.

As shown in Figure 2, the main system on the SGI Onyx interacts with the GUI application on the SGI Indy via multicast networking protocols. Although multicasting is inherently an unreliable means of network communication, it provides some attractive features. Namely, it allows multiple Soldier Station suites to co-exist on the same physical network by partitioning the network traffic into separate multicast groups which each suite can subscribe to (Pratt et al., 1996a). The ability to subscribe to multiple multicast groups, if desired, also allows the future development of a logger application which can record multicast network traffic for user analysis purposes.

4. System Development

The integration of two distinct simulation systems, Janus and NPSNET, was much easier said (and drawn) than done. The integrated system was envisioned to work together seamlessly, however, the two component systems are based on radically different modeling philosophies, are written in different computer languages, and utilize different terrain file formats. The integration was primarily carried out in an stepwise manner with often multiple iterations occurring at each step:

```
do
  merge code
  test results
do
  debug problems
  test results
until CORRECT
until DONE
```

4.1 Integrating Janus and NPSNET

Integration of the two modules which comprise the main Soldier Station system required careful coordination and consideration between the VM and CM developers. The first step was to identify exactly what the types of interactions were sought between the VM and CM. This step produced five main CM driver routines as listed in Table 2 (Ohman, 1996).

4.1.1 Modifications to Janus

The appropriate constructive model algorithms for DIs were isolated from the original Janus code. The CM continues to use the exact same input data files as Janus (i.e. FORCE, DEPLOY, JSCRN, and SYSTEM) and has complete knowledge of all entity attributes and system characteristics. These CM routines control one interactive DI entity so at least one system type in the FORCE file must be the same as the Soldier Station system type specified during program startup. The CM must inform the VM of the DI entity system type's capabilities in order for the user to be able to effectively control the entity. Thus, SS_setup passes back the names of the weapons and sensors along with the starting number of ammunition rounds available for each program run.

Some modifications to the Janus routines were introduced to resolve time and space incompatibilities and to provide enhanced user control. Since Janus is an event driven simulation, changes were made to allow the CM algorithms to be called in real-time and re-

ardless of the frame rate. Built-in time delays for the soldier's movements due to obstacles and suppression by fire were shortened from minutes to seconds in order to allow the user to respond in a realistic amount of time to such situations. All references to nodes for routes and movement control were removed with input now being provided by the Soldier Station operator via the Flybox. The DI entity may move in one direction and look in another by having his head turned. He may also now move backwards so that the soldier can remain facing forward while retracing his last steps instead of having to turn around. He can also now enter any infantry foxhole or vehicle prepared fighting position regardless of which Janus side it belongs to.

Unlike traditional Janus DIs, the Soldier Station entity can fire his weapon on command at no specific targets (e.g. generate suppressive fire), and also fire in a non-horizontal plane. He can detect up to twenty-five targets (instead of just ten as for Janus DIs) provided he is alive and does not have a major wound which might impact his ability to detect targets. If targets are detected, the CM allows the Soldier Station to fire at them regardless of side assuming that other firing criteria have been met. Thus, the Soldier Station entity can always possibly commit fratricide if he does not exercise good judgment in the synthetic battlefield. As in Janus, the firing criteria which must simultaneously be met include meeting minimum safe range requirements for firing a weapon, and meeting weapon jam/clearing times and ammunition reloading times. Short firing delays are introduced if any of these criteria are not met.

Along with added control the user is also faced with more potential simulation hazards. For example, the entity is now also able to step on and detonate detected mines whereas Janus entities can not detonate mines previously detected. Automatic defilade status changes have also been removed so that the DI entity no longer changes from fully exposed to partial defilade when he stops moving unless directed by the user to do so. In fact, the DI entity may remain fully exposed while being fired upon if the user does not take any action.

For demonstration purposes, the DI entity is also allowed to be resurrected if killed. In addition to possibly being killed or suppressed, the soldier may also now be wounded. For determining a wound status, the body is divided into six parts, each having a probability of being wounded depending upon the current posture. The damage assessment routine processes direct fire, indirect fire or mine explosion events.

4.1.2 Modifications to NPSNET

To integrate the CM with the VM, NPSNET program flow was examined to determine where and how the CM driver routines should be called. Since Soldier Station was designed to use the NPSNET visual simulation framework, the relatively minor modifications were needed to be able to interact with them. These included making coordinate conversions to/from NPS coordinates to Janus UTM coordinates and transferring information from C++ dynamic data structures into static arrays to pass to the CM. Code was also added to map internal NPSNET vehicle numbers of remote entities to Janus unit numbers. These Janus unit numbers are pre-defined in the Janus FORCE file which is read when SS_setup is called during system startup. Remote DIS entities must pass an appropriate Janus unit number in DIS Entity State PDU markings fields to the VM in order to be recognized by the CM as valid units (a requirement which will hopefully be removed in the near future).

Strict interfaces for each of the CM driver routines were defined, e.g. function name and the number, type and order of the arguments. C++ wrappers (which account for the C++ name mangling of function names and facilitate correct argument type passing) were then created so that the CM functions could be called directly from the VM. The five main CM driver routines and numerous other supporting routines are archived together in a library object and linked to the VM.

There were several features added to NPSNET to comply with the additional capabilities required for Soldier Station. To allow the Soldier Station entity the ability to select between up to five different weapons, additional weapon models (besides the existing M16 rifle) were added with only one showing at any given time. Additional sensor views for the binocular and gun sights sensors were added as 2D overlays over the 3D view along with concurrent changes in the field of view. To receive the user's inputs from the UIS, multicast IDUs were defined.

Support for up to six different sides of DIs was provided. This involved creating DI models with numerous different colored uniforms and one model which carried no weapon (to be used as a civilian). Low resolution soldier models were also incorporated for low level detections which assume that features such as uniform patterns, faces, objects carried and even limbs are not visible. Presently, only DIs have multiple levels of detail models available.

4.2 Conversion of Janus Terrain

The terrain data formats used by Janus and visual systems such as NPSNET are completely different. The terrain elevations in Janus are gridded and assumed to be constant within each grid square (or pixel based). If represented in 3D graphics, the terrain would appear as a collection of cubes with different heights. On the other hand, in NPSNET the terrain file format is polygonal in nature and its representation consists of triangles connecting each grid point to the next grid point forming a relatively smooth terrain surface when compared to Janus' terrain representation.

The relatively small size of the DI entities demands smooth 3D terrain representations in order to avoid large visual abnormalities in the terrain database at the edge of each Janus grid square. However, the CM routines still represent the terrain as gridded cells internally. Thus, there is a mismatch in the internal terrain representation for each module which sometimes causes the DI entity to appear either above or below the actual polygonal ground surface. Some detection mismatches would also likely occur causing entities to disappear/appear although the VM may not actually show terrain blocking/not blocking the line of sight. Both of these problems are more predominant in terrain areas where steep and/or rapidly varying gradients exist.

As part of the NPSNET-Janus integration, a SGI-based software tool was developed to convert the Janus terrain into a MultiGen Flight format for the VM. The terrain tool reads in the Janus gridded terrain elevations and the separate polygonal Janus feature data (vegetation, roads, rivers, buildings, etc.) and converts them into the required polygonal MultiGen format. Since the CM uses the Janus terrain format while the VM uses the polygonal database format, the Soldier Station system relies on this ability to convert various Janus terrain databases. With the availability of MultiGen terrain databases, other DIS-compatible simulations which also use the format are also able to interoperate with Soldier Station. So far, Soldier Station has successfully participated in numerous exercises with remote DIS entities generated from Janus linked to DIS (JLINK) (Pate and Roussos, 1996), NPSNET, and another Soldier Station system.

The terrain tool is capable of automatically handling most types of Janus terrain databases with minimal user intervention. Because of the very large number of grid points present in most Janus terrain databases, the tool subsamples the points, using every

other grid point, in an effort to reduce the number of polygons. The time it takes to convert a terrain database depends strongly on the number of Janus polygonal terrain features as well as on the number of elevation grid points present. A medium sized terrain database (say, 10 km by 10 km) with an average number of polygonal features takes about ten minutes to convert. Currently, small tree areas with relatively large concave edges can cause some conversion problems.

4.3 Main Program Flow

Figure 3 is a simplified program flow chart of the integrated NPSNET-Janus main system. Essentially, various initializations are carried out and then several major tasks are carried out continuously as long as the simulation continues. These tasks include the handling of DIS network PDUs, handling user input IDUs, simulation of the local DI entity, updating the position and statuses of the remote entities and finally drawing the scene. It is noted, however, that usually network management and drawing would be handled asynchronously in a multiprocessing mode.

During program startup, the VM calls SS_setup once. Various Janus data files are read and, if desired, post processor data files are started to log data from the Soldier Station during the simulation. In the simulation loop, PDUs from other remote entities on the DIS network are processed. User input IDUs from the GUI are processed similarly, and given the current inputs, SS_move is called to determine the soldier's next position and his movement status. Remote entities are updated using dead reckoning, and once a second SS_search is called to determine what live entities the Soldier Station entity can detect. Dead entities are not detected by Janus and are not passed to SS_search. However, they are still displayed on the GUI 2D map as black icons and appear visibly damaged on the synthetic battlefield.

When an IDU packet contains data about a trigger pull effected, SS_reload is called to determine the outcome of the firing event. If the weapon was successfully fired, a fire PDU is sent out over the DIS network. If a close proximity detonation PDU is received from another remote site, SS_assess is called to determine the extent of an injury, if any. If the soldier is uninjured, the simulation proceeds as before. If the soldier is killed, he can be resurrected by the user in demo mode, or the user can elect to exit the simulation. If the soldier is killed or wounded, movement, detection and/or firing capabilities will be impaired depending on the injury. For simplicity, Figure 3 assumes that the DI entity is uninjured.

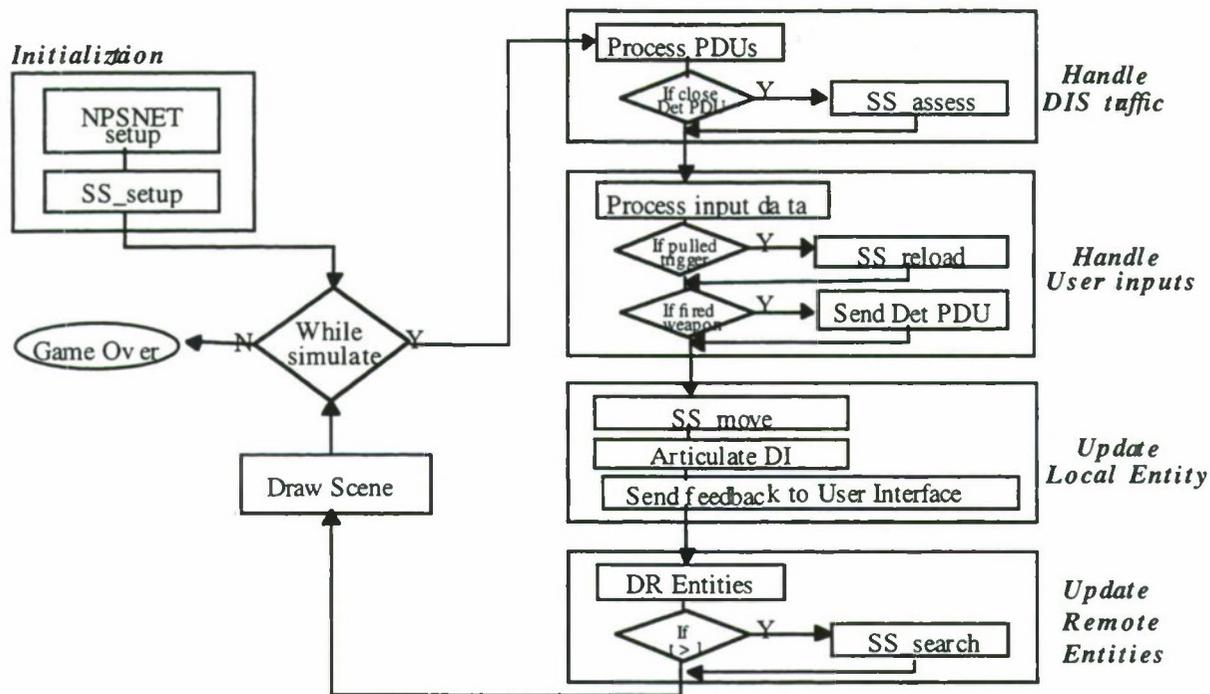


Figure 3: Simplified Main Program Flow

During each frame, feedback data about the current status of the DI entity and what entities have been detected at what detection level (aimpoint, recognition and identification) is sent back to the GUI. This feedback information is then displayed on the GUI and could affect the user's next inputs. Table 3 lists some of the possible outcomes from making function calls to the CM driver routines.

The program flow shown in Figure 3 is actually nearly the same as the normal program flow of NPSNET with the exception of the calls to CM routines. Without the CM driver routines, NPSNET assumes that the DI entity can always move regardless of terrain characteristics, can detect anything that is drawn, can fire upon anything when the trigger is pulled, and is always fatally wounded by any close proximity detonation. Clearly, the integration of the Janus algorithms brings much needed realism into the visual simulation.

5. Conclusions

The decision to merge Janus and NPSNET together to form one seamless Soldier Station system was, to a large extent, governed by practical reasons. Namely, the analysis needs of the Soldier Station project could

be met while significantly reducing project development time and costs by integrating two existing systems rather than developing either one independently. By reusing code from NPSNET, Soldier Station acquired a major head start on underlying 3D graphics, basic entity simulation, DIS networking and sound requirements. New development efforts could be focused on the integration with Janus, adding necessary features which were not currently available in NPSNET and the development of the UIS. By merging Janus algorithms, the virtual simulation acquired robust mobility characteristics and target detections as well as realistic weapon firing outcomes and injury assessments. These features replaced non-existent or very simplistic (and generally unrealistic) graphics based capabilities which were previously available in NPSNET.

The combined system, represents a significant improvement over either of its component parts, but it does have some limitations. Currently the representation of terrain, soldier movements and engagements, and visual parameters are at moderate levels of detail. These are subject to change according to the resolution needed by the simulation, but large, high resolution terrain databases with many entities (say, more than fifty) present will degrade the system performance without further optimizations (as mentioned below). The inherently different VM and CM inter-

nal terrain representations causes some visual inconsistencies which need to be resolved. For improved DIS-compatibility, unit numbers should be able to be created or assigned dynamically within the CM. The transfer of data from VM's dynamic data structures to static arrays for the CM routines is unavoidable without major code changes to the VM data structures. However, this could impact performance otherwise and requires some in-depth system performance analyses beforehand. Some data transfers are simply unavoidable due to C++ and FORTRAN language differences.

For increased system performance, we plan to spawn a separate process to obtain the computationally expensive CM detection routine output asynchronously via shared memory buffers. Optimizations to the visual simulation include an upgrade to SGI Performer 2.x which supports terrain database paging and increased use of level of detail modeling techniques. To significantly lower hardware system costs, we plan to tune the main system to run on the new, much less expensive SGI Maximum Impact workstations. Enhancements for Soldier Station to participate in night time and urban environment simulations are also planned in the near future.

6. Acknowledgments

The authors would like to thank Mr. David Ward, CPT Steve Brown, CPT Bill Smith, MAJ Glen Roussos, Mr. David Hastings, LTC Ralph Wood, COL Carl Baxley (Ret) and Mr. Roy Reynolds for their help and valuable inputs to the system. Development and demonstrations of the system would not have been possible without the support of TRAC-WSMR, TRAC-MTRY and NPS computer systems personnel and students.

7. References

- Granieri, J. and Badler, N. (1995). Simulating Humans in VR. To appear in R. Earnshaw, J. Vince, and H. Jones, editors. *Applications of Virtual Reality*, Academic Press.
- Naval Postgraduate School, Computer Science Department (1996). NPSNET Research Group Internet Home Page <http://www-npsnet.cs.nps.navy.mil/npsnet>.
- Ohman, D. (1996) *Soldier Station Combat Module Documentation*. Prepared for TRAC-WSMR by Nations, Inc.
- Pate, M., and Roussos, G. (1996) JLINK - A Distributed Interactive Janus. *Phalanx*, Vol. 29, No.

I, 12-15.

- Pratt, D., Barham, P., Barker, R., and McMillan, S. (1996a) AUSA 95 DI Demonstration. *14th DIS Workshop Proceedings*, 1165 - 1170.
- Pratt, S., Pratt, D., Waldrop, M., Barham, P., Ehlert, J., and Chrislip, C. (1996b) Humans in Large-scale, Real-time, Networked Virtual Environments. Submitted for publication in *Presence*.
- Rohlf, J., and Helmann, J. (1994). IRIS Performer: A High Performance Multiprocessing Toolkit for Real-Time 3D Graphics. *SIGGRAPH '94 Proceedings in Computer Graphics*, 381-394.
- U.S. Army TRADOC Analysis Center, White Sands Missile Range, NM (1996) Janus Version 6.0 Documentation.

8. Authors' Biographies

Shirley Pratt is a Computer Scientist in the Computer Science Department at Naval Postgraduate School (NPS). She is the lead developer of the NPSNET visualization module for the Soldier Station system. Ms. Pratt has a M.S. in Ocean Physics from the U.C. San Diego and a B.A. in Applied Mathematics from U.C. Berkeley. Her research interests include real-time modeling of the environment, virtual simulations of dismounted infantry, and the use of efficient DIS aggregation protocols.

CW4 David Ohman (Ret) is an Operations Research/Systems Analyst working for Nations, Inc. in support of the Soldier Station project for TRADOC Analysis Center, White Sands Missile Range (TRAC-WSMR). Prior to joining Nations, he was the only military member assigned as a Software Developer for the Janus Interactive Simulation Development Division at TRAC-WSMR. He has an M.S. in Industrial Engineering from the University of Texas, El Paso, an M.A. in Management from Webster University, and a B.S. in Business and Management from the University of Maryland. He is also a graduate of the US Army Operations Research/Systems Analysis Military Applications course.

David Pratt is serving as the first Technical Director of the Joint Simulation System (JSIMS) Joint Project Office in Orlando, Florida. He holds this position concurrently with an appointment as a tenure track faculty member at the Department of Computer Science, NPS. Prior to joining the faculty at NPS, Dr. Pratt was a Data Processing Officer in the United States Marine Corps. He holds a Ph.D. and a M.S. in Computer Science from NPS and a B.S. in Electrical Engineering from Duke University. His research interests include distributed simulation and

architectures to support scalability in real-time 3D computer graphics.

for constructive models for the past four years. He has a B.S. in Civil Engineering from New Mexico State University.

John Galloway is an Operations Research Analyst for TRAC-WSMR. Since joining TRAC-WSMR in 1986, he has been involved with Combined Arms Support Task Force Evaluation Model (CASTFOREM), Janus and Soldier Station. He is the Program Leader for the Soldier Station project and has worked within the dismounted infantry arena

Control Item	Available Options	User Interface
Posture	Upright, Crouching, Kneeling, Prone, Fox hole, Deploy weapon, Align head and body orientation, Lase target	Flybox Buttons
Sensor	Eye balls, Binoculars, Gun Sights (Thermal in future)	GUI Radio Buttons
Weapon	M16A2 Rifle, M203 grenade launcher, M60 machine gun, M249 semi-automatic weapon, M72 light anti-tank weapon (Object Individual Combat Weapon in future)	GUI Radio Buttons
Hand Signals	Various signals for movement control, formations, fire control, emergency alerts, echelon designation, and other miscellaneous signals	GUI Push Buttons
Map Display	Map displayed, not displayed Zoom in, zoom out Show topographical contour shading, grid lines, terrain features, buildings, obstacles Detected entity icons shown, not shown	GUI Push Buttons

Table 1: User Selectable Options for Soldier Station

CM Routine	Purpose	Example Considerations
SS_setup	Reads Janus data. Passes DI capabilities back to VM	Scenario, run, system number inputs
SS_move	Determines current location, actual speed, movement status	Soldier posture / orientation, terrain characteristics, requested speed, suppression status, wound status
SS_detect	Determines what live entities are visible at what detection level	Soldier posture / wound status, active sensor, target defilade status/speed, LOS probability
SS_reload	Determines firing result, impact point, rounds remaining	Soldier posture / orientation, target type, active weapon, rounds left, time last fired
SS_assess	Determines injury, if any, due to a close proximity detonation	Soldier posture, munition type, impact location, firing entity, luck

Table 2: Description of the Five CM Driver Routines

CM Routine	Basic Outputs	Specific Information
SS_move	Movement status	Moving on a road, in vegetation, in an urban area, in a river
	Speed status	Moving at requested speed, slowed by terrain, moving at maximum speed allowed, not moving because soldier is kneeling or in a fox hole
	Delay status	Obstructed by a building, fence, another unit, by a river, an abatii, a smoke pot, a mine
	Posture status	In the requested posture, not in a fox hole because none nearby
SS_detect	No entities detected	None
	Entities detected	For each entity, detection at level: aim point = 1, recognition = 2, identification = 3
SS_reload	Successfully fired	Fired at a target, or generated suppressive fire Impact point is XYZ
	Unsuccessfully fired	Unable to fire because: Soldier is wounded, moving too fast, being suppressed, not ready to fire Range is too far, too close Weapon is out of ammunition, pointed at too large of a pitch angle Target is a non-combatant, is dead, a low probability hit, a bad target, an identified friendly target
SS_assess	Injury status	Not injured, dead, wounded
	Wound type	Hit in the head, chest, stomach, pelvis, leg, arm
	Suppression status	Not suppressed, suppressed by fire

Table 3: Example Feedback Data from theCM

Session 4a: Learning

Adamson, DERA, UK

Hieb, GMU

Rajput, UCF/IST

Fogel, Natural Selection, Inc.



Genetic Algorithms and Force Simulation

J. Adamson
Centre for Defence Analysis (Land)
DERA Fort Halstead
Sevenoaks
Kent TN14 7BP
United Kingdom

Dr K. G. Joshi
EDS Defence Ltd
Centrum House
101 - 103 Fleet Road
Fleet, Hampshire GU13 8NZ
United Kingdom

1. Abstract

Realistic modelling of force behaviour in a combat simulation or wargame increases in difficulty and importance with the complexity of the simulation. Where there are many activities available to the modelled entities and a large quantity of information on which decisions will be based, the need for realistic modelled behaviour without the need for excessive user intervention is at its greatest.

The Centre for Defence Analysis (CDA) and EDS have explored the possibility of using Genetic Algorithms (GAs) as a means of developing realistic tactical behaviour within a detailed combat simulation, CDA's Close Action Environment (CAEN).

CAEN is a highly detailed model of the close combat battle. It is both a means of simulating weapons effects and an interactive wargame between opposing forces of up to platoon level strength. Up to 200 entities are usually modelled and consist of either infantrymen and their personal weapons or vehicles such as armoured personnel carriers and main battle tanks. The representation of tactical activities within CAEN makes it possible to apply the GA mechanisms of evolution and selection to entity behaviour in an effective manner.

The study work undertaken shows that GAs can succeed in developing feasible behaviour as a consequence of straightforward primary goals such as individual survival and the achievement of tactical objectives. The results also show that the effectiveness of the GA method is closely linked to the fidelity of the underlying combat model, and is likely to produce increasingly realistic behaviour as the simulation environment itself becomes more complex.

2. Background

2.1 Modelling Realistic Behaviour

Complex computer simulations of the battlefield require not only the details of the military hardware, but also the tactics and behaviour of the entities involved. The problem of the detailed modelling of realistic tactics is the subject of current research by the Centre for Defence Analysis, DERA Fort Halstead.

A major limitation of existing simulations is the inability of models to represent subjective decisions and to provide common-sense, realistic behaviour in a range of situations. This limitation applies both to fully autonomous simulations and to wargames which are driven by interactive commands from users. It is particularly evident in high-resolution models, since the greater the detail and realism of the underlying simulation, the more complex is the decision making faced by the simulated entities. For example, a high fidelity model such as CAEN provides a large number of activities which an entity may choose and a large quantity of information on which the entity may base its decision.

2.1 Involvement of the User

A problem often encountered with wargames is the high level of user effort required to drive them. To avoid overloading the players with detail, wargames are usually equipped to enable commands to be issued to groups of entities which then move and engage in the same way. This is broadly realistic except in situations when the success of the group depends upon the differences between its entities rather than their similarities. For example, in advancing under fire, it is advantageous for some entities to supply covering fire while their comrades advance.

2.3 Intelligent Entities

For detailed aspects of wargames such as the above, adequate realism can only be achieved by intensive user involvement. If a method can be found to define a degree of intelligent response by computer generated entities to general instructions, then the user is freed to control the overall picture or to observe and analyse. In addition, the level or expertise required by the user can be reduced, the potential complexity of the behaviour can be increased and the realism of the behaviour can be matched to the underlying model.

The last point is particularly important. As models increase in complexity, more intelligent decisions must be made, and more of these will have to be taken by the system rather than the user.

3. Use of Genetic Algorithms

3.1 Optimisation

Intelligent computer generated entities must be able to respond to general instructions in a manner which is both effective and realistic.

In combat models, if we can represent the elements of entity behaviour in a suitable manner, we may hope to obtain realistic responsive behaviour as a result of optimising these elements with respect to the achievement of military goals.

The effectiveness of this approach will depend on the selection of a suitable optimisation technique. Genetic Algorithms (GAs) provide a robust approach to optimisation with a number of features relevant to this application.

3.2 Genetic Algorithms

Genetic Algorithms (Goldberg 1989) have been applied to a number of optimisation problems in complex domains. The GA uses principles based on genetic processes in nature. It operates by maintaining a population of diverse solutions to a problem, combining elements from members of this population, and selecting the fittest from which to form the next generation. In this way it iteratively seeks stable optimal solutions, with inefficient paths being rejected at each iteration.

The method is effectively a parallel one in that the variety of the paths provide a greater chance of overcoming obstacles to the optimisation, and the

discarding of unsuitable candidates at each stage narrows the search. The advantages of this approach include:

- The population domain need not be continuous.
- Wide coverage of the domain of possible behaviours is maintained at all times.
- The solution will not converge to a local maximum with poor global performance.

3.3 Principal Features

The principal features of GAs are:

- The features of the system to be optimised are encoded in a data series, called a chromosome. The encoded features are referred to as genes.
- Fitness is defined by means of an objective function defined on each chromosome.
- A population of chromosomes, or candidate solutions, is maintained and modified in a series of iterations, or “generations”.

This leads to an iterative process which should eventually converge to a population of highly performing individuals. The algorithm will then be terminated by a suitable criterion.

The iteration is governed by three operations:

- Reproduction.
- Recombination.
- Mutation.

3.3.1 Reproduction

The fitness of each chromosome determines its probability of selection for the next generation, causing the better chromosomes to dominate. In this study, a weighting based on fitness ranking was used to discard the worst individuals in favour of the same number of newly created individuals.

3.3.2 Recombination

Pairs of chromosomes may “mate” and reproduce by exchanging genes, enabling the creation of individuals combining the best features of both. The techniques used in this study were:

- “Uniform crossover”, where the values of the gene in each location for two randomly selected parents may be exchanged according to a probability test.
- “Average crossover”, where the selected gene pairs are replaced by their average value.

3.3.3 Mutation

Random changes may cause particular genes in a child chromosome to differ from those of its parent. A variant (random creep) was also used, in which random increments were added to the (real number) values of randomly chosen genes.

4. CAEN

4.1 Summary of CAEN

The CAEN model was chosen to investigate the use of GAs in force simulation.

CAEN is a two-sided close combat model representing entities down to the resolution of individual infantrymen with their personal weapons or armoured vehicles. It can be used either as an automatically replicated simulation with no user intervention, or as an interactive wargame in which players create and control their own forces within a terrain database, supported by graphical displays. At the CDA, CAEN is used for operational analysis and weapon system evaluation.

CAEN provides a highly suitable environment for the assessment of GAs (or other tactical optimisation methods), on at least two counts:

- The level of modelling resolution is detailed enough to represent the tactical options available to individual entities in complex scenarios.
- The parametric method used to specify tactical behaviour provides an appropriate basis for the application of the GA mechanism.

4.2 Tactical Behaviour

The tactical behaviour available to entities in CAEN is very flexible. It is defined in terms of simple base activities such as changes in posture and speed, surveillance, target acquisition, direct fire or suppressive fire. Each entity has access to a variety of information about its current situation and it can use this data to make decisions about which activity to perform next.

A key feature of CAEN is the control of this behaviour by input data files set up by the user prior to the game. The files, known as Activity Sequences, consist of a sequence of predefined activities together with test conditions to allow branching to different points in the sequence. Activity sequences thus provide tactical algorithm templates parametrised by

quantities such as the time spent performing an activity and the conditions for entering and leaving the activity. Random elements are provided to avoid the behaviour of an entity becoming too predictable.

Changes to entity behaviour in the simulation are effected by editing one or more activity sequences. Group tactics, executed by all entities within a group, are contained in Tactics files which reference a set of activity sequences. In this way, hard-coded rules of behaviour are replaced by data files allowing behaviours to be easily created and modified by setting parameters.

Although the mechanics of tactical definition are easy, the creation of an effective and realistic activity sequence for even a simple scenario is a complex and time consuming problem. Considerable care must be taken to ensure that the resulting behaviour is as desired. This is where the use of Genetic Algorithms provides a potentially valuable means of improving fidelity and automating the process of tactical definition.

5. Application of Genetic Algorithms to CAEN

5.1 General Approach

The application of GAs to CAEN is based on representing Activity Sequences as chromosomes. Several sets of parameters are randomly generated and used to define tactical algorithms. These are then used as input data by the CAEN simulation, using a scenario suited to the tactical template. For example, if a minefield is specified in the scenario, then the tactical template should include a mine clearing activity.

The results of the CAEN simulation are then processed to determine the effectiveness of each activity sequence, using a suitable measure of effectiveness (MOE). In the studies to be described, the MOE is based on a combination of simple military criteria, namely the proportion of Blue forces killed and the time taken to reach the objective. A number of CAEN replications are carried out with each chromosome/activity sequence in order to build up a significant MOE. A new generation is then formed by the procedures of Section 3.3, and the process iterated until a reasonably stable tactical algorithm is established.

5.2 The Study Environment

The study environment made use of an existing GA harness previously developed by EDS for the

optimisation of target recognition rules. The GA harness was linked to CAEN by an interface program which translated parametric data between the CAEN activity sequence format and the GA manipulations. A main control program scheduled the repeated runs of the GA harness and the CAEN replications.

5.3 Modifications to CAEN

The CAEN functionality was augmented to allow entities to develop behaviour based on two new areas of information, namely:

- Awareness of own forces.
- Awareness of surroundings (terrain and culture).

These features had not been needed in previous uses of CAEN where the tactics of advancing troops were preordained by the user. This is in contrast to awareness of enemy forces, and the effect of terrain and culture on detection and engagement, which are fully represented and used in executing these tactics.

6. Studies and Results

The use of Genetic Algorithms for tactical development within CAEN was assessed by carrying out a series of three studies using tactical activity sequences of increasing complexity. These were:

1. An undetected advance.
2. An advance under fire.
3. A “generic” advance in which the tactical sequence is largely unspecified.

6.1 Undetected Advance

6.1.1 Activity Sequence

In this first study, a simple tactical activity was chosen to enable the GA principle itself to be assessed. The main objects were:

- To verify that tactical optimisation was achievable.
- To assess the effects of the parameters governing the optimisation process.

An Undetected Advance activity sequence was defined to represent a group of infantry moving towards an objective in a straight line. While moving forward the infantry may or may not engage in suppressive fire, and may change speed and posture. If an entity acquires a target, it engages in direct fire.

The sequence thus consists of two phases of different durations:

- Direct fire only.
- Suppressive and direct fire.

On entering one of these phases, the entity will make a posture/speed decision, randomly selecting one of run, walk, crawl or remain in the same speed/posture as before. This activity sequence can therefore be parameterised by allowing the variation of the following data items:

- The duration of each phase.
- The probability of selecting each posture/speed on starting the phase.

This results in eight numbers: one time and three probabilities for each phase. These eight numbers are the values to be optimised.

6.1.2 Scenario

A simple scenario was defined with a small number of Blue and Red infantrymen armed with rifles. The Blue objective was defined as a particular building occupied by Red forces. The distance to the objective was defined such that the Blue entities would need to repeat their tactical activities several times. This helped to ensure that the optimised parameters were not too closely tailored to the specifics of the scenario.

6.1.3 Measure of Effectiveness

The success S of a run was defined by the formula:

$$S = (1 - L) \exp(-t/t_0)$$

where L is the proportion of Blue forces killed, t is the time taken to reach the objective, and t_0 a time constant of the order of a typical value of t . This fitness function was used in all three studies.

6.1.4 Results

The study used a population of ten chromosomes, initially selected at random. The progress of the optimisation between generations was monitored though a 10 generation moving average of the fitness measure S for the most successful chromosome.

The study runs were broadly successful. The fitness values of the chromosomes used were found to increase as the evolution progressed, and the individual genes were found to converge.

One outcome was to highlight the effect of the stochastic nature of the CAEN simulation on the optimisation process. As stated earlier, CAEN performs a number of statistical replications to estimate the mean fitness value for each chromosome. It is more difficult for the GA system to distinguish the fittest chromosomes for the next generation when this estimate is inaccurate due to random effects. Increasing the number of replications had the twofold effect of speeding the GA optimisation and making the results more precisely defined. About ten replications per chromosome were necessary to produce effective optimisation in this case.

Despite these limitations, the resulting activity sequence showed an improvement on *a priori* parameter estimates of the type that would be adopted in a normal use and a large improvement on randomly chosen parameter values.

6.2 Advance under Fire

6.2.1 Activity Sequence

This study aimed to optimise a more complex activity sequence, where the Blue entities come under significant suppressive fire. The previous activity sequence was enhanced to enable entities to make the decision, when encountering suppressive fire, to either engage immediately or to attempt to avoid hostile entities by moving to another position. In order to design an activity sequence with this degree of flexibility, a much larger number of parameters, which included times, probabilities, search ranges and speeds, needed to be fixed. A suitable subset of these was chosen for genetic evolution, giving a chromosome length of 17.

6.2.2 Scenario

The previous scenario was enhanced by providing the Red infantry with heavy suppressive fire. The Blue attacking force was much larger. This was intended to represent overwhelming force, so that even fairly poor activity sequences would be capable of completing the mission. This allowed non-zero fitness values to be associated with all sequences, which in turn allowed the ranking of all chromosomes within a generation.

6.2.3 Results

Considerable difficulty was found in optimising this activity sequence. Many adjustments had to be made by the sequence designer to obtain any success in the scenario. The optimised sequence was observed to be inferior to the Undetected Advance.

The lesson learnt from this study was the importance of correct design of the tactical template on which the GA operates. For example, the design assumption of attempting to avoid enemy fire is preferable to simply returning fire, appeared to be flawed. More generally, the GA system was unable to make progress with a complex sequence where it had little control over the ordering of activities. The GA system was unable to improve what transpired to be essentially a poor design.

6.3 A Generic Sequence

6.3.1 Activity Sequence

This study had the objective of taking a very general structure for the activity sequence which effectively allowed the GA to determine which activities should be used in the scenario.

In defining a generic sequence, it was noted that any activity sequence could be divided into three types of component, namely:

- Base activities.
- Fixed subsequences of base activities.
- Decision tests.

Following execution of any of these activities, the generic sequence will jump to a chromosome defined successor, which may be another decision test or the entry point to another fixed subsequence.

In this way, each chromosome defines a pattern of execution of activity subsequences and decision tests.

As many possible decision tests and activity subsequences were made available. Contrary to the previous studies, every effort was made to ensure that the activity parameters were not chromosome driven. This was done to avoid unnecessarily long chromosomes as increasing the chromosome length would increase the run time required to perform the optimisation.

The chromosome eventually used had 44 genes.

6.3.2 Scenario

The underlying scenario was similar to that in the previous studies. No fixed route was pre-defined for the Blue entities; the Blue entities would have "Entity Routes" defined dynamically by the optimised activity sequence.

6.3.3 Results

The generic sequence was successfully optimised with virtually no need for modification to the initial sequence design. The optimisation was more straightforward than for either of the earlier sequence types. This was a clear consequence of deliberately avoiding tactical assumptions at the design stage.

The GA process was allowed to shape the tactics. The resulting activity sequence did not exploit all the potential decision tests and activities. Instead, it concentrated on the two fundamental activities of moving towards the objective and engaging the enemy.

The Blue entities evolved the tactic of destroying Red entities as a means of achieving the primary goals of low casualty rate and speed of reaching the objective.

7. Conclusions

The studies reported in this paper have successfully demonstrated the feasibility of using Genetic Algorithms for the automatic generation of tactical behaviour in detailed combat simulations such as CAEN. It has been shown that using the GA process to optimise effectiveness in terms of clear primary objectives can result in the evolution of tactical behaviour that helps to achieve those goals.

It has been seen that the method is potentially most powerful when given the most scope in evolving tactical sequences. Attempting to constrain the algorithm *a priori* to a poorly designed tactical template may be worse than useless. But given freer range, the method is capable of converging quickly to an effective tactical sequence.

The study has also shown how the efficiency of the method is closely linked to the accuracy of the underlying simulation. For example, using larger simulation runs to improving statistical accuracy benefits the convergence of the GA and may improve overall efficiency.

Achieving the goal of tactical realism for computer generated forces through this approach will depend ultimately on the fidelity of the combat simulation itself. The nature of the evolutionary process is such that the evolved behaviour will adapt closely to its environment, and the Genetic Algorithm can only work within the constraints imposed by the model itself.

Future enhancements to the fidelity of CAEN, such as improvements in terrain modelling and psychological factors, may help to demonstrate a corresponding improvement in tactical realism.

It is precisely in the context of increasing model complexity, where the demands on the user become most severe, that the Genetic Algorithm approach seems to offer the most potential.

8. Acknowledgements

The investigation which is the subject of this paper was initiated by Land Studies Department, Centre for Defence Analysis, DERA Fort Halstead, Sevenoaks, Kent, TN14 7BP, and was carried out under the Terms of Contract No. CDA/H/131.

9. References

Goldberg, David E. (1989). *Genetic Algorithms in Search, Optimisation and Machine Learning*. Addison-Wesley.

10. Authors' Biographies

Janusz Adamson is a Senior Consultant at the Centre for Defence Analysis, DERA Fort Halstead. Mr. Adamson has a BSc(Hons) degree in Astronomy and an MPhil. His project responsibilities include Command Agent Support for Unit Movement (CASUM), the Close Action ENvironment wargame (CAEN), Genetic Algorithms, Real-time Knowledge base Systems and Command Agents. His technical focus is on Synthetic Environments and Computer Generated Forces.

Dr Keith Joshi was employed for three years by EDS Defence Ltd's Research and Studies Group as a Systems Engineer. A Mathematical Physicist with a doctorate in Theoretical Physics Dr Joshi specialised in combat modelling during his career with EDS.

Training a ModSAF Command Agent Through Demonstration

Michael R. Hieb, Gheorghe Tecuci, J. Mark Pullen
Department of Computer Science
George Mason University,
Fairfax, VA 22030
{hieb, tecuci, mpullen}@cs.gmu.edu

1. Abstract

As Computer Generated Forces (CGF) technology advances to where Command Forces (CFORs) are constructed and deployed, automated knowledge acquisition tools will become increasingly important. Since CFORs are expected to emulate human behavior, their development will require more knowledge acquisition than previous CGF efforts, however knowledge acquisition has traditionally constrained the development of knowledge-based systems. This paper presents a ModSAF command agent called Virtual Commander (VCDR) that the Subject Matter Expert (SME) can "teach" using ModSAF editors. VCDR is built upon Agent-Disciple, a multistrategy apprenticeship learning system that provides machine learning and knowledge acquisition methods in a "toolkit". With VCDR, an SME gives the CGF command agent specific examples of problems and solutions, explanations of these solutions, and supervises the agent as it solves new problems, all through the ModSAF interface. We have prototyped this training approach with the Captain system, that allowed an SME to teach a ModSAF company commander how to defend its assigned area of responsibility. In this paper we describe the design of VCDR, the learning and problem solving algorithms it utilizes, and novel prototype implementations of both a distributed interface that integrates learning functions (Agent-Disciple) to CGF (ModSAF), as well as graphical ModSAF editors for VCDR.

2. Introduction

The ability to build intelligent command agents for CGF is significantly constrained by the knowledge acquisition effort required. Many iterations by SMEs, programmers and knowledge engineers are required to develop acceptable behavior even for a narrow range of situations. Moreover, once built the agents cannot adapt themselves to changes. Various automated knowledge acquisition tools have been proposed and utilized for this problem, but there is no standard acquisition methodology for CGF that has gained acceptance. The existing approaches primarily utilize programmers and knowledge engineers to encode the expertise of a SME. Our goal is to have the SME use a familiar simulation interface to

instruct a CFOR agent directly. This direct instruction reduces the involvement of programmers and knowledge engineers, increasing the efficiency of the acquisition process and improving the quality of the acquired knowledge.

VCDR agents are instructable ModSAF agents, providing a new approach to solving the knowledge acquisition problem for CFORs. VCDR follows a general methodology for developing instructable agents for existing applications given in Hieb (1996). VCDR utilizes Programming by Demonstration (PDB) (Cypher, 1993) and Machine Learning techniques to allow instruction by an SME. Programming by Demonstration systems give an end user the ability to create programs by demonstrating their actions thorough a graphical user interface. This is a new research area that is concerned with interactive learning of user tasks from a limited number of examples and explanations given by the user. Machine Learning uses more formal, domain-independent autonomous learning methods. Often the input to machine learning programs are either large numbers of examples, extensive background knowledge, or, for multistrategy learning systems, both.

In our approach, an SME teaches a VCDR agent through the ModSAF Graphical User Interface rather than using a different interface for the learning system. The SME initially demonstrates to the VCDR agent how to perform a new mission. The SME uses the existing ModSAF task editors to "program" the agent, as the SME normally would, creating a sequence of specific tasks. This is given as an initial example of the mission to the learning system. The SME then explains the relevant features of the mission. The learning system will then attempt to perform a different instance of the mission (e.g. on a different piece of terrain) under the supervision of the SME, asking the SME to classify its solution of the mission as a correct or incorrect example. The SME uses ModSAF's graphical user interface to correct the agent if it does not perform the mission as required by the SME. After this teaching session, the VCDR agent will have learned how to perform this type of mission (i.e., create a rule specifying how to select tasks and instantiate task parameters for a specific mission) and be able to perform this new

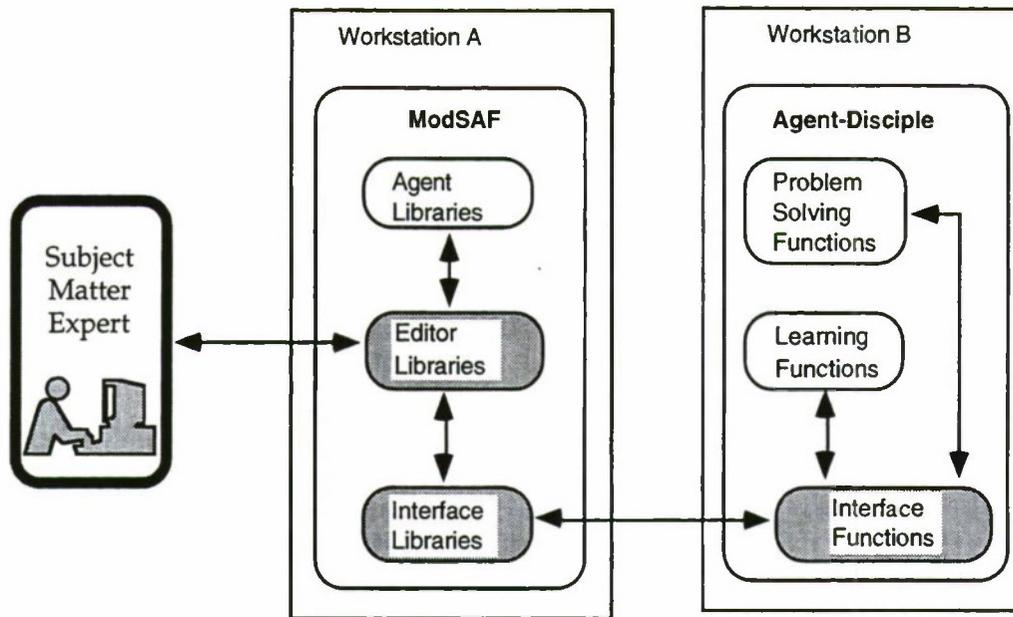


Figure 1: VCDR Design

mission when asked to do so by the SME, without requiring the SME to program the behavior.

We have prototyped this approach with the Captain system (Hieb, 1996; Hieb & Tecuci, 1996; Hieb et al. 1995) which consisted of an integration of the apprenticeship learning system, Disciple (Tecuci 1988), and ModSAF. In Captain, a ModSAF company commander can be taught how to place its platoons to defend its assigned area of responsibility. This process involves eliciting an initial example from the SME, eliciting 5 to 10 explanations and showing the SME 5 to 10 examples of solutions that the system generates. Experiments with Captain indicate that the system will scale up as it is applied to learning other tasks in this domain. Captain utilized the ModSAF terrain map to show examples to the SME for classification, but did not fully integrate other learning interactions (specifying examples and explanations) into the ModSAF interface.

Figure 1 shows the system design of VCDR. An SME uses editors within ModSAF to teach a command agent new tasks. The machine learning is performed on a separate workstation running Agent-Disciple software (Agent-Disciple provides the learning functions of Disciple in a modular toolkit). The gray modules indicate modules that are being modified or created for VCDR.

In order to fully integrate VCDR with ModSAF we are expanding upon the Captain interface in two areas. We are developing a series of ModSAF editors

that will provide an integrated interface for the instruction process, rather than using the interface of the learning system. For instance, the new ModSAF VCDR editors allow the SME to use the terrain map interface during the explanation process, rather than requiring the use of a textual interface. Also we are interfacing the learning functions (which are Lisp-based) to ModSAF using experimental protocol data units (PDUs).

The remainder of this paper is organized as follows. Section 3 presents an extended discussion of related work. Section 4 presents the special format of VCDR rules. Section 5 describes the implementation of VCDR, including a prototype interface between Disciple and ModSAF, and the design of ModSAF editors designed for agent training. Finally, Section 6 concludes the paper with a discussion of our agent-building approach.

3. Related Work

We first review some of the related research on learning, particularly Apprenticeship Learning and the new field of Programming by Demonstration. Then we describe how this research applies to agents.

3.1 Learning

Apprenticeship Learning systems are at the intersection of the fields of Machine Learning and Knowledge Acquisition. An Apprentice Learning System can be defined as an interactive knowledge-based consultant that is provided with an initial

domain theory and is able to assimilate new problem-solving knowledge by observing and analyzing the problem-solving steps of its users through their normal use of the system (Tecuci and Kodratoff 1990).

Apprenticeship Learning systems involve the user in the learning process, where Machine Learning systems generally are not interactive. In Apprenticeship Learning systems the user provides the learning system's input in a representation that is natural to the user. The learning system has an interaction with the user during the learning process, where the user may be asked to give other examples, confirm a hypothesis, or give explanations. The output of this learning is generally presented to the user prior to being translated into a form usable by a performance element (e.g., a rule-based production system) (Tecuci & Hieb, 1994). Most Machine Learning systems require their learning input to be put into a special format. The user may not be able to understand the input (which may be in the form of data) or the output (which is often in the form of rules) unless the user is quite familiar with the learning method.

Knowledge Acquisition systems and Apprenticeship Learning systems are closely related. However, the emphasis of most Knowledge Acquisition systems is on modeling the initial knowledge base and eliciting knowledge. The emphasis in Apprenticeship Learning systems is on refining knowledge that has already been elicited or created, using machine learning techniques that learn from a user.

PBD systems give an end user the ability to create programs by demonstrating their actions. Machine learning covers an overlapping area of research concerned with methods that learn concepts from example or domain theories, including such instruction from a teacher. An example of a PBD system is the Metamouse system (Maulsby & Witten, 1993) gives the user the ability to automate drawing tasks. The user instructs an agent (a turtle named Basil) on how to manipulate objects through an innovative graphical interface. Basil learns from specification of graphical constraints to construct a program that automates graphical editing. The program can have loops and conditionals.

PBD systems are significant because their goal is to empower the end user by assuming that, if a user knows how to perform a task on the computer, then that knowledge should be sufficient to create a program to perform the task. Rather than learn a programming language, the user should be able to instruct the computer to watch as the task is demonstrated (Cypher, 1993). A common concern among these systems is interface design. The graphical (and

verbal) user interface of these systems is generally very sophisticated, and the inferencing techniques are usually more specific to the task domain than machine learning methods (Maulsby, 1994).

PBD systems generally deal with the automation of simple tasks. They generally do not deal with automating complex tasks or behaviors, such as concerns the ModSAF agents. The systems do not provide facilities for the end user to specify domain knowledge to the system, as is done with knowledge elicitation or knowledge acquisition methods.

3.2 Instructable Agents

Software agents are programs that can execute with their own identity within an application, either autonomously or semi-autonomously. The agents that currently are being developed either have fixed (non-adaptive) behavior or can exhibit some limited forms of learning. Agent-Disciple can be thought of as an agent development environment either for training existing agents or for building entirely new agents.

ModSAF agents use a task-level architecture similar to a subsumption architecture. This allows a user to give orders to an agent, who then attempts to carry out the orders, unless it reacts to a condition for which it was programmed (e.g., a threat). Since this is a reactive architecture, the agents must be supervised closely by the SME.

Other approaches have been used to develop entirely new ModSAF agents using the SOAR problem-solving model (Tambe et. al, 1995). These agents currently operate primarily in air environments. The Soar-based agents have the potential to significantly improve the behavior of ModSAF entities, but conducting the knowledge acquisition to build such agents remains a difficult problem.

Soar (Laird, Newell & Rosenbloom, 1987) is a general problem-solving architecture that addresses the problem of agent learning. SOAR has a learning mechanism that is integral to its architecture-chunking. In contrast, Agent-Disciple integrates many learning methods for agent instruction. Soar has a very elaborate model of problem solving – the Problem Space Computational Method (PSCM) – that uses deductive rules. By Contrast Agent-Disciple uses rules with plausible conditions and is able to reason with incomplete knowledge, although its problem solving model is also more complicated than most expert systems.

Huffman (1994) used Soar in his system Instructo-Soar, where an instructable agent. learns from tutorial

instruction. Instructo-Soar learns general knowledge from specific instructions, using rote learning and a type of inductive learning (situated explanation) in addition to the chunking of SOAR. Huffman delinates the type of knowledge that must be learned to build an agent in SOAR's computational model, and demonstrates the ability of Instructo-Soar to acquire the majority of knowledge types necessary. Instructo-Soar has been demonstrated in a small domain, a blocks world with a small number of operators, properties and relationships (less than 10 of each).

4. Plausible Version Space Rules in VCDR

We first describe the novel structure of rules in VCDR and then give examples of how VCDR uses such *plausible version space* rules.

4.1 Plausible Version Space Rules

VCDR uses a hybrid knowledge representation integrating semantic networks and rules. Semantic networks represent information from a terrain database at a conceptual level, as well as knowledge about forces and weapon systems. In order to facilitate learning, the objects and the rules both use the following representation unit:

```
(concept-i concept-k (FEATURE-1 value-1)
...
(FEATURE-n value-n))
```

This expression defines 'concept-k' as being a subclass of 'concept-i' (from which it inherits

features) with additional features. The value of a feature may be a constant or another concept.

In VCDR, rules are procedures that consist of a PROBLEM statement, CONDITIONS and a SOLUTION statement. Each condition (also called a *clause*) consists of a plausible upper bound and plausible lower bound which are in the format of the representation unit described above. The plausible upper bound is a conjunctive expression that is supposed to be more general than the exact condition, and the plausible lower bound is a conjunctive expression that is supposed to be less general than the exact condition. The two bounds define a *plausible version space* (PVS) for the condition to be learned by Disciple (Tecuci, 1992). The bounds and the version space are called plausible because the learning process takes place in an incomplete representation language that may cause them to be inconsistent (a lower bound that covers some negative examples or an upper bound that does not cover all positive examples).

Figure 2 shows the general form of a PVS procedure in VCDR. A procedure is learned from specific problem solving episodes indicated by a user. Once learned, a procedure can be selected to be performed by an SME. A *mission* is a goal specification given to the agent (the agent in the military simulation is given an order). A *task* is an action that the agent can take in the simulation.

The terms p_1 through p_n and p_{11} through p_{iv} represent parameter names, m_1 through m_n represent mission parameters, t_{11} through t_{iv} represent task

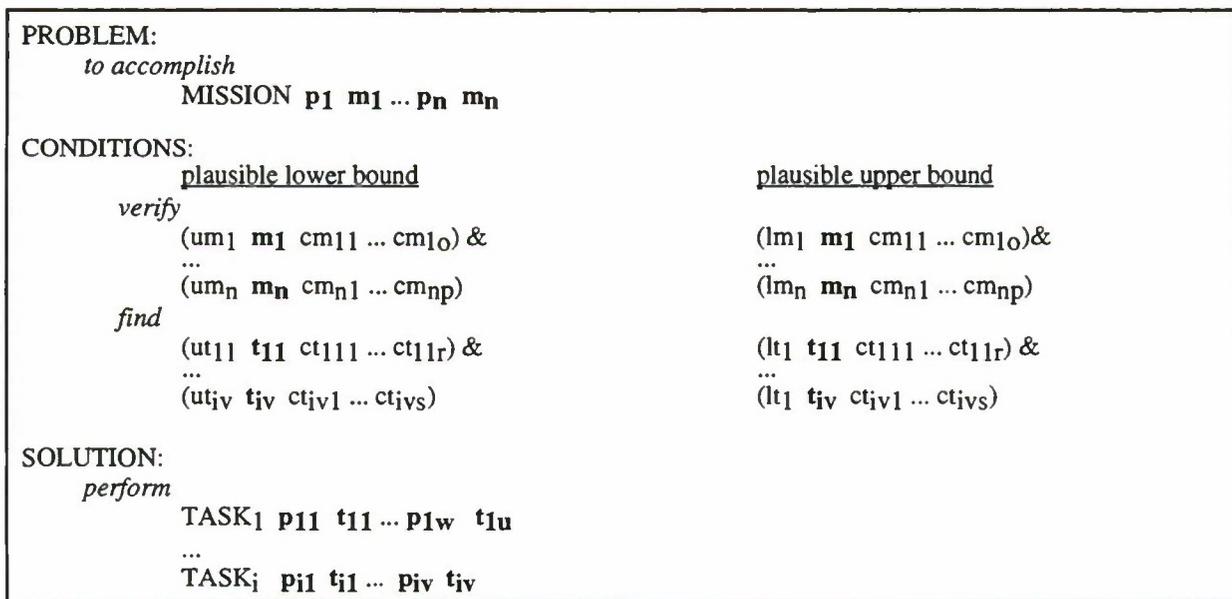


Figure 2: PVS Procedure

parameters, um and ut represent the upper bound class, lm and lt represent the lower bound class, and cm through ct represent constraints upon the parameters. The plausible upper bound and plausible lower bound are both conjunctive expressions. The plausible upper bound is more general than the plausible lower bound. The upper bound represents a set of possible solutions, while the lower bound represents the least general generalization of the set

of solutions actually encountered.

4.1 Examples of Using PVS Rules

Figure 3 contains simple procedures for an agent corresponding to Figure 4. Procedure P1 specifies how to OBSERVE an object t1: verify that it meets the constraint in the lower bound – that it is a terrain-element (both the upper and lower bounds are the

P1:	
<i>to accomplish</i>	
OBSERVE TERRAIN t1	
<u>plausible lower bound</u>	<u>plausible upper bound</u>
<i>verify</i>	
(terrain-element t1)	(terrain-element t1)
<i>find</i>	
(hill t2 (OPPOSITE t1))	(hill t2 (OPPOSITE t1))
(armored-platoon u)	(platoon u)
<i>perform</i>	
MARCH UNIT-ID u LOCATION t2	
<i>with the positive examples</i>	
(t1 " hill-60-70, t2 " hill-44-91, u " platoon-a2)	
(t1 " lake-57-82, t2 " hill-60-70, u " platoon-a2)	
<i>with the negative examples</i>	
(t1 " hill-44-91, t2 " lake-57-82, u " platoon-a2)	
(t1 " hill-60-70, t2 " hill-44-91, u " company-a)	
P2:	
<i>to accomplish</i>	
MOVE UNIT-ID c LOCATION t	
<u>plausible lower bound</u>	<u>plausible upper bound</u>
<i>verify</i>	
(company c (COMMANDS p1)	(company c (COMMANDS p1)
(COMMANDS p2)	(COMMANDS p1)
(COMMANDS p3))	(COMMANDS p3))
(hill t)	(terrain-element t)
<i>find</i>	
(armored-platoon p1)	(platoon p1)
(armored-platoon p2)	(platoon p2)
(infantry-platoon p3)	(platoon p3)
<i>perform</i>	
MARCH UNIT-ID p1 LOCATION t	
MARCH UNIT-ID p2 LOCATION t	
MARCH UNIT-ID p3 LOCATION t	
<i>with the positive examples</i>	
(t " hill-60-70, c " company-a, p1 " platoon-a1, p2 " platoon-a2, p3 " platoon-a3)	
(t " hill-44-91, c " company-h, p1 " platoon-h7, p2 " platoon-h8, p3 " platoon-h4)	

Figure 3: PVS Procedures

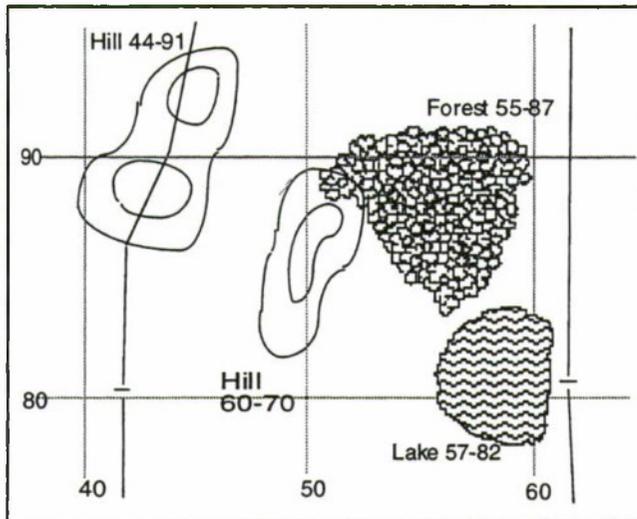


Figure 4: Terrain Map

same in this case); if it is, then find objects for the task parameters t_2 and u subject to the constraints in the lower bound – that t_2 is a hill opposite from t_1 and that u is an armored-platoon; if there are no armored platoons, then use the upper bound and attempt to find a platoon; if objects for the task parameters are found, then perform the march task.

Procedure P1 has been learned from the following initial example.¹

```

to accomplish
OBSERVE
  TERRAIN hill-60-70
perform
MARCH
  UNIT-ID platoon-a1 LOCATION hill-44-91

```

The initial example is expressed as a tuple,

(t_1 " hill-60-70, t_2 " hill-44-91, u " platoon-a2)

A detailed description of how the upper and lower bounds are formulated and modified to obtain P1 is given in (Hieb, 1996). The second example,

(t_1 " lake-57-82, t_2 " hill-60-70, u " platoon-a2)

is positive and indicates that the lake in Figure 3.1 can be observed from hill-60-70 by a platoon. The example

(t_1 " hill-44-91, t_2 " lake-57-82, u " platoon-a2)

¹ An instructor gives the initial example and classifies the subsequent examples in this scenario as positive or negative. This scenario focuses on the learning method rather than interaction.

is negative since the platoon cannot move onto the lake (the platoon is a motorized unit with tracked vehicles and cannot drive on the lake). The example

(t_1 " hill-60-70, t_2 " hill-44-91, u " company-a)

is negative since a company cannot be utilized as the observing unit (it is too large to perform the observation mission).

After the procedure is learned, it can be used by the agent as follows:

- 1) SELECT – The agent selects a procedure to accomplish a specific mission and binds the variables in the problem to the mission parameters.
- 2) VERIFY – Verify that the mission parameters meet the constraints imposed by the corresponding *verify* lower bound conditions.
- 3) FIND – Find a set of objects corresponding to the task parameters that meet the constraints in the *find* lower bound conditions.
- 4) EXECUTE – Instantiate the task(s) in the solution with the set of objects from steps 2 & 3 corresponding to the parameters of the task(s), and invoke the task(s).

If the agent cannot find a procedure in step 1 to accomplish the mission, or the mission parameters do not meet the constraints imposed by the *verify* lower bound conditions in step 2, or the agent is unable to find a solution in step 3 (a set of objects meeting the constraints in the *find* lower bound conditions), then the agent will be unable to accomplish the mission. To simplify the problem solving, only the lower bound is used. The upper bound is manipulated during learning and is kept so that the rule can be modified later.

For example, the user of the simulation may wish to have the agent monitor for enemy activity in the area depicted by Figure 4. The user selects the agent and orders it to OBSERVE forest-55-87.

- 1) SELECT – The agent selects P1 to OBSERVE forest-55-87 and binds the variable t_1 to the object forest-55-87.
- 2) VERIFY – The agent checks that the object represented by parameter t_1 meets the constraints imposed by the *verify* lower bound condition in P1:

(terrain-element t_1)

Since forest-55-87 is a terrain-element, **t1** is verified.

- 3) FIND – Find the objects corresponding to **t2** and **u** that meet the constraints in the *find* lower bound in P1:

(hill **t2** (OPPOSITE **t1**))
(armored-platoon **u**)

t1 was bound to the object forest-55-87 in step one. The object found for **t2** must be a hill opposite from the object represented by **t1**. This must be hill-60-70 according to the knowledge in the agent's semantic network. Then the object found for **u** must be an armored platoon. platoon-a2 is found, but could be any other armored platoon.

- 4) EXECUTE – The agent orders platoon-a2 to execute the march task to hill-60-70.

Similarly, procedure P2 has been learned from the following initial example:

to accomplish

MOVE

UNIT-ID company-a LOCATION hill-60-70

perform

MARCH

UNIT-ID platoon-a1 LOCATION hill-60-70

MARCH

UNIT-ID platoon-a2 LOCATION hill-60-70

MARCH

UNIT-ID platoon-a3 LOCATION hill-60-70

This procedure specifies how to move a company to a position – by moving each of the platoons associated with that company using the appropriate movement task for a platoon (MARCH). The relationship COMMANDS must hold since otherwise a company could “take” another company’s platoons. There are two positive examples, and the procedure is less completely learned than procedure P1. For example, the variables representing the platoons in the lower bound could be further generalized from armored-platoon and infantry-platoon to platoon, since it does not matter what type the platoons are.

When performing procedure P2, there is an additional complication since the constraints on mission parameter **c** involve other variables representing task parameters from the solution. In this case a least commitment strategy is used, where it is merely verified that the object represented by **c** has the relationship COMMANDS to three other objects in the VERIFY step. These objects are then found in the FIND step.

5. Implementation of VCDR

The design of VCDR follows the Agent-Disciple methodology of creating an Agent Training Environment from (Hieb, 1996). The basic learning and problem-solving functions were taken from the Agent-Disciple toolkit. In addition to using the core methods in Agent-Disciple’s toolkit the following are required: 1) translators between ModSAF’s data structures and the semantic network; 2) implementation of an interface between the learning functions written in Lisp and the ModSAF libraries written in C; 3) constructing ModSAF training editors; 4) integrating the new tasks learned into the existing task-level architecture; and 5) modifying the task editor so that the new tasks can be selected. The overall implementation is depicted in Figure 5.

We discuss our approach to 2) and 3) below.

5.1 Distributed Interface to Learning System

As researchers in the field of Programming by Demonstration have found, it is very difficult to interface learning systems to existing applications. We have designed a distributed interface, since the two systems are quite different. This also has the advantage of allowing the use of a separate CPU (from that running ModSAF) to run the learning functions. To convey the data from the ModSAF training editors, we are interfacing the learning functions of Disciple to ModSAF using an experimental Distributed Interactive Simulation protocol data unit (PDU).

In designing this interface, we distributed control of learning between ModSAF and the learning functions in Lisp. ModSAF is in control of sending PDUs to Disciple. The Lisp process blocks while waiting to receive PDUs. When Disciple receives a PDU, it processes the message part, then sends a PDU back to ModSAF in reply (either carrying data or sending an acknowledgment) and blocks.

Table 1 shows the interface protocol created for the interface. The phases correspond to distinct sets of learning functions as in Hieb (1996). Within the phases, types are discrete events, triggering actions (PDUs sent activating the learning functions in Lisp). Even types indicate ModSAF sending PDUs to Disciple, while odd types indicate Disciple sending PDU to ModSAF.

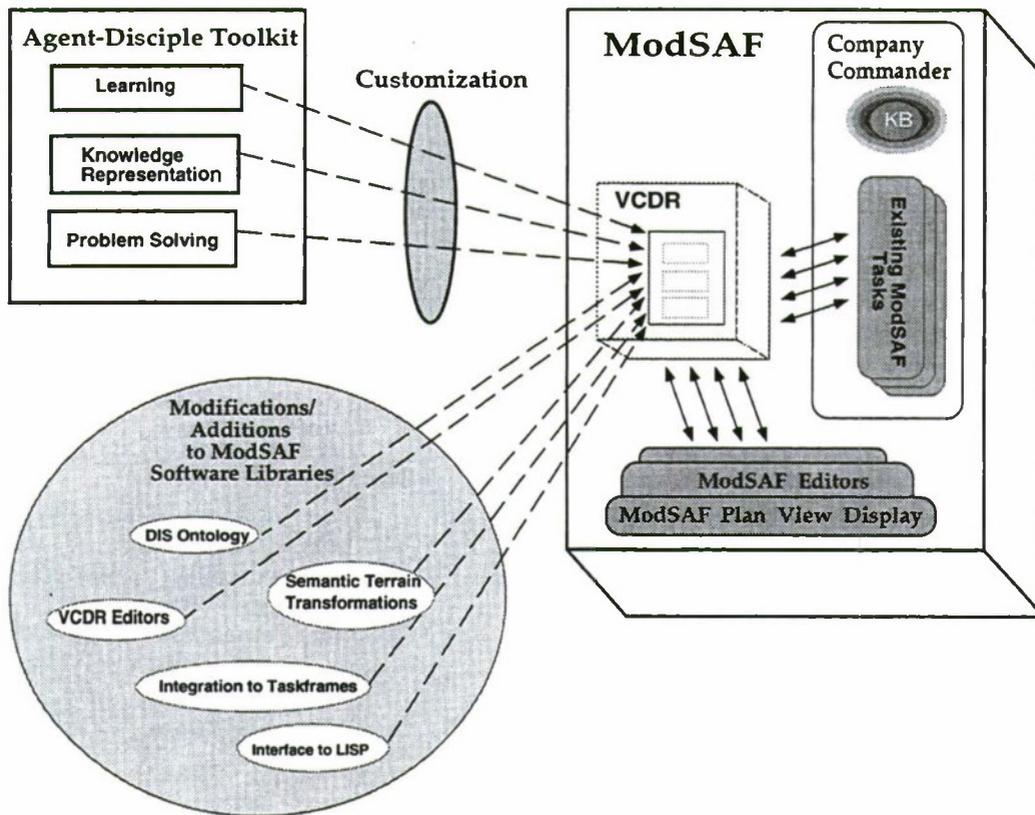


Figure 5: Constructing VCDR Using the Agent-Disciple Toolkit

Specification of Initial Scenario Phases	
Phase 0 - Give Initial Example	Phase 4 - Generate Experimentation Example
Type 0 - Select Example Template	Type 0 - Request Example to be Generated
Type 1 - Send Acknowledgement of Template	Type 1 - Send Example
Type 2 - Select Initial Example	Phase 5 - Give Experimentation Example
Type 3 - Send Acknowledgement of Initial Example	Type 0 - Specify Example
Phase 1 - Give Initial Explanations	Type 0 - Send Acknowledgement of Example
Type 0 - Select Variable to Generate Explanations from	Phase 6 - Classify Experimentation Example
Type 1 - Send List of Explanations	Type 0 - Classify Example
Type 2 - Select Explanation(s)	Type 1 - Send Acknowledgement of Classification
Type 3 - Send Acknowledgement of Explanations	Phase 7 - Explain Mistake in Experimentation
Phase 2 - Quit Initial Signal Phases	Type 0 - Select Variable to Generate Explanations from
Type 0 - Quit Specification of initial	Type 1 - Send List of Explanations
Type 1 - Send Acknowledgement of Quit	Type 2 - Select Explanation(s)
Learning through Experimentation Phases	Type 3 - Send Acknowledgement of Explanations
Phase 3 - Experimentation Search Parameters	Type 4 - Select Variable to Blame
Type 0 - Select Variable(s) to Fix	Type 5 - Send Acknowledgement
Type 1 - Send Acknowledgement Fixed	Type 101 - Quit Explanation Phase
Type 101 - Quit Search Phase	Type 102 - Send Acknowledgement
Type 102 - Send Acknowledgement of Quit	Phase 8 - Quit Experimentation Signal Phases
Type 102 - Send Acknowledgement of Quit	Type 0 - Quit Experimentation
	Type 1 - Send Acknowledgement of Quit

Table 1: Protocol for Agent-Disciple/ModSaf Interface

The specific implementation of the PDU interface functions in both ModSAF and Disciple is covered in much greater detail in (White et. al. 1996).

5.2 VCDR ModSAF Editors

A major part of building a VCDR instructable agent is developing the graphical user interface for the SME instructor. We have designed a series of ModSAF editors to act as the agent instruction interface. A ModSAF editor is a GUI that allows the user to directly manipulate ModSAF data structures. The VCDR editors allow an SME to give both examples of how to perform a mission and also explanations to the VCDR agent, using the existing ModSAF interface. During the teaching session the VCDR agent will learn the mission and later, be able to perform it when asked to do so.

A challenge for the interface designer of a new ModSAF editor is to use as much of the existing ModSAF interface as possible, including both the terrain map (or plan view display) and the editables within the existing ModSAF editors. For VCDR we are designing an interface within ModSAF that both utilizes elements of the plan view display (such as the command and control graphics) and new text editables to allow a user to specify examples and explanations during learning.

An analysis of existing interactive learning systems shows that there are at least four main classes of interaction necessary to teach an instructable agent (Hieb, 1996): knowledge specification, specification of training examples, specification of explanations, and classification of examples. We have designed and are implementing editors that support these interactions.

Knowledge specification is mainly performed during the construction of the agent, prior to the phase when the SME instructs the agent. However, the SME will specify different terrain areas to utilize as training and testing data to the learning system. Thus a VCDR terrain editor will allow the SME to both draw boundaries around the terrain area and also guide the learning system in performing semantic terrain transformations (Hille et. al. 1995), a form of semi-automated knowledge acquisition.

MITRE is developing a Command and Control Simulation Interface Language (CCSIL) to provide CFORs a common language for command and control, utilizing military terms and message formats (Salisbury et. al. 1995). Since an agent instruction interface must utilize predefined knowledge about the mission and terrain, some CCSIL language constructs can be used as standardized terms during instruction.

The remaining classes of interactions are implemented in the VCDR instruction editors, which provide the capability to select command and control graphics from the plan view display. Figure 6 shows the Main VCDR editor, which allows an SME to specify the initial task to be learned ("Start"), to begin the training process ("Train"), to verify the task taught ("Verify") and to assign an agent the new task ("Use"). Figure 7 shows the Training Editor, which is invoked when the "Train" button is pressed in the Main Editor. The Training Editor allows the SME to construct training examples for the system ("Give Examples") or to have VCDR generate examples (through experimentation) for the user to classify. Classification of training examples is provided via a yes/no/unknown menu option button. Figure 7 shows the Verification Editor, which is invoked when the "verify" button is pressed from the Main Editor. This editor allows the user to verify a rule by selecting another terrain area and specify variable assignments of a task.

All of the editors also provide a history of past examples and explanations via buttons on a common utility pane, as well as providing the user to directly examine the rule and variable assignments.

7. Conclusions and Future Research

Systems for automating complex tasks must be designed so that they can be general enough to be adapted to different domains. For example, considerable effort was expended in modifying both the ModSAF application (which contains over 450 source libraries written in C) and Agent-Disciple to create Captain. The next goal, in VCDR, is to use the existing editor interface of ModSAF, as opposed to a separate learning system interface. Lieberman (1994) points out that the interface between an end user and the agent training system is a crucial issue not addressed in most of the machine learning research. The VCDR approach is to use as much of the existing ModSAF interface as possible, on the assumption that this is easier for the SME.

The VCDR instruction method requires a pre-existing knowledge base and the creation of customized methods to translate the application's current state to the learning function's semantic network. To address this drawback in the ModSAF domain, terrain transformation techniques have been developed and are being implemented to automatically create a substantial portion of our semantic network from the digital terrain databases (Hille et al 1995). Also, facilities are provided for the SME to specify additional terms in the representation language during the training process, as in Dybala & Tecuci (1995).

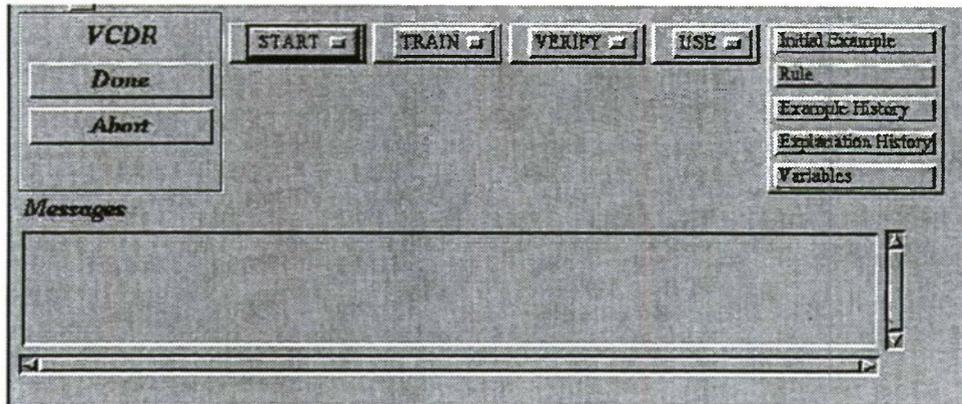


Figure 6: VCDR Main Editor

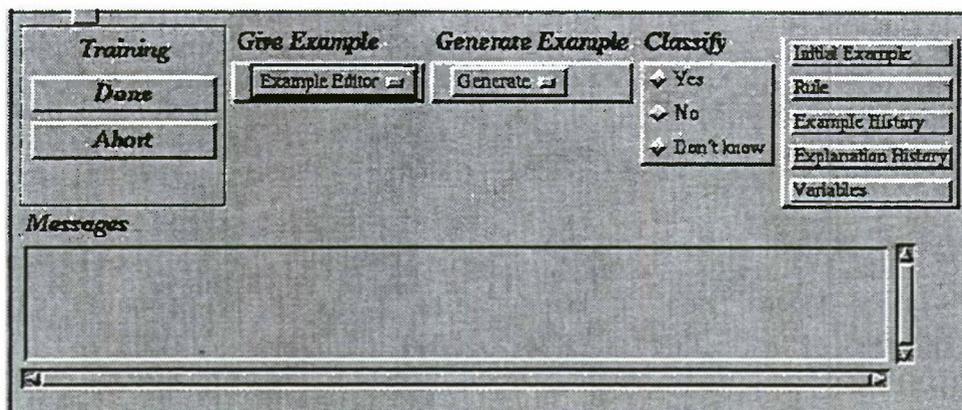


Figure 7: VCDR Training Editor

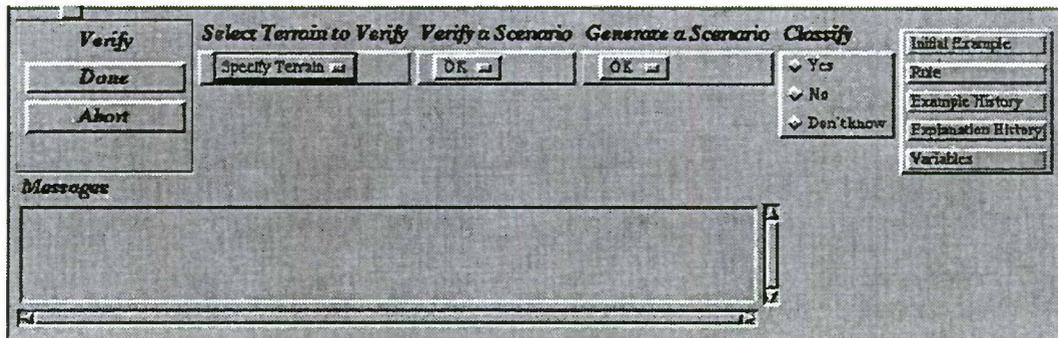


Figure 8: VCDR Verification Editor

From our experience we have concluded that it is difficult to give the user the flexibility to define completely new complex tasks such as missions in the ModSAF domain. The missions that a ModSAF agent can perform are quite complicated because the environment is complex and non-deterministic. In VCDR we provide the SME a task template corresponding to the basic missions available to the SME. The user can then specialize or modify this template to create the task structure necessary for learning a procedure.

Much of the power of the agent instruction approach presented comes from the multiple types of interaction between the SME and the agent being taught. Such rich interaction is rare among Machine Learning systems, and is closer to the interaction found in Programming By Demonstration systems (Maulsby, 1994). Such interaction is necessary, however, to develop more powerful agents. These interactions include: specifying new terms in the representation language of the agent; giving the agent an example of a solution to a task for which the agent

is to learn a general procedure; validating analogical instances of solutions proposed by the agent; explaining to the agent reasons for the validation; and being guided to provide new terms in the representation during interaction (Tecuci & Hieb, 1994).

VCDR addresses the basic requirements for an ideal Programming By Demonstration learner, as identified by Maulsby and Witten (1995). First, the learning agent is under the user's control, who specifies the actions and features relevant to the task to be taught, gives hints and explanations to the agent, and guides its learning actions. Second, the learning agent uses various knowledge-based heuristics for performing plausible generalizations and specializations that are understandable, including plausible generalization of a single example. It also learns from a small set of examples. Third, the agent learns a task in terms of all the parameters necessary for task execution.

VCDR does not currently address autonomous learning, where the agent would learn without the guidance of an SME, but the same learning methods that are being developed for instruction should be applicable (Hieb, Hille & Tecuci, 1994; Hille, Hieb, & Tecuci, 1993; Tecuci et. al., 1994).

Verification and validation is a difficult problem with CFORs, because of the complexity of the agent reasoning process. Our approach addresses this problem by allowing the user to test the agent with additional examples after the agent has successfully learned how to perform a mission. The SME can select the testing examples or the testing examples can be automatically generated. If the agent performs the mission incorrectly, the user can correct the agent through the same instruction techniques that were originally used to teach the agent (i.e., by giving additional examples or explanations). If the agent performs the mission selected by the SME correctly, then confidence in the learned behavior increases.

VCDR offers an efficient approach for teaching complex behavior to an agent through demonstration. This approach was illustrated by our investigations with the Captain system (Hieb, 1996). This approach to training ModSAF agents appears to be more natural and significantly simpler than the currently process, where the SME manually specifies the mission of the ModSAF agents in great detail to achieve reasonable behavior in a simulation. The learning efficiency in VCDR is achieved through the use of plausible version spaces and a human guided heuristic search of these spaces.

8. Acknowledgments

This research was conducted in the Center for Excellence in Command, Control, Communications and Intelligence and the Computer Science Department at George Mason University. Work on ModSAF was sponsored in part by DMSO under DISA contract DCA100-91-C-0033 and work on Disciple was sponsored in part by Advanced Research Projects Agency Contract No. N66001-95-D-8653. The authors thank Ken Frosch of the C3I Center for designing/implementing the Disciple/ModSAF PDU interface, Jeffrey Sullivan of the U.S. Army Topographic Engineering Center for developing the prototype VCDR editors, and Vince Laviano of the C3I Center for ModSAF programming.

9. References

- Ceranowicz A., (1994). ModSAF Capabilities. *4th Conference on Computer Generated Forces and Behavior Representation*, May, Orlando, Florida.
- Cypher, A. (Ed.). (1993). "Watch What I Do: Programming by Demonstration," *MIT Press*, Cambridge, MA.
- Dyballa T. and Tecuci G. (1995). Shared Expertise Space: A Learning Oriented Model for Cooperative Engineering Design. In *Proc. IJCAI-95 Workshop on Machine Learning in Engineering*. August. Montreal, Canada.
- Hieb, M.R. (1996). *Training Instructable Agents Through Plausible Version Space Learning*, PhD Dissertation, School of Information Technology and Engineering, George Mason University, Fairfax VA. (<http://cne.gmu.edu/~hieb>).
- Hieb, M.R. and Tecuci, G. (1996). Training an Agent through Demonstration: A Plausible Version Spaces Approach. *Proceedings of the 1996 AAAI Spring Symposium on Acquisition, Learning and Demonstration: Automating Tasks for Users*. AAAI Press Technical Report, Menlo Park, CA.
- Hieb, M.R., Tecuci, G., Pullen J.M., Ceranowicz A., & Hille D. (1995). A Methodology and Tool for Constructing Adaptive Command Agents for Computer Generated Forces. *Proceedings of the 5th Conference on Computer Generated Forces and Behavioral Representation*. May. Orlando, Florida.
- Hieb, M.R., Hille D. and Tecuci, G. (1993). Designing a Computer Opponent for War Games: Integrating Planning, Learning and Knowledge Acquisition in WARGLES. In *Proceedings of the 1993 AAAI Fall Symposium*

- on Games: Learning and Planning, AAAI Press Technical Report FS-93-02, Menlo Park, CA.
- Hille D., Hieb M.R., Pullen J.M. & Tecuci G. (1995). Abstracting Terrain Data through Semantic Terrain Transformations. *5th Conference on Computer Generated Forces and Behavioral Representation*. Orlando, Florida.
- Hille D., Hieb, M.R. & Tecuci, G. (1994). Captain: Building Agents that Plan and Learn. In *Proceedings of the 4th Conference on Computer Generated Forces and Behavioral Representation*.
- Huffman, S.B. (1994). Instructable Autonomous Agents. *PhD Thesis*. Department of Computer Science and Engineering. University of Michigan.
- Laird, J.E., Newell, A. & Rosenbloom P.S. (1987). SOAR: An Architecture for General Intelligence. *Artificial Intelligence*. Vol. 33.
- Lieberman, H. (1994). A User Interface for Knowledge Acquisition From Video. In *Proc. Eleventh Conference on Artificial Intelligence*, Morgan Kaufmann.
- Maulsby, D. (1994). Instructable Agents. *PhD Thesis*. Department of Computer Science. University of Calgary.
- Maulsby, D. and Witten, I.H. (1995). Learning to Describe Data in Actions. *Proc of ICML-95 Workshop on Learning from Examples vs. Programming by Demonstration*, CA.
- Salisbury, M.R., et al. (1995). "Implementation of Command Forces (CFOR) Simulation," *5th Conference on Computer Generated Forces and Behavioral Representation*.
- Tambe, M., Johnson, W.L., Jones, R.M., Koss, F., Laird, J.E., Rosenbloom, P.S. & Schwamb, K. (1995). Intelligent Agents for Interactive Simulation Environments. *AI Magazine*. 16(1), Spring.
- Tecuci, G. (1988). DISCIPLE: A Theory, Methodology and System for Learning Expert Knowledge, Ph.D. Thesis, University of Paris South.
- Tecuci G. (1992). "Automating Knowledge Acquisition as Extending, Updating and Improving a Knowledge Base," *IEEE Transactions of SMC*. 22(6).
- Tecuci, G. & Hieb, M.R. (1994). Consistency-driven Knowledge Elicitation: Using a Machine Learning-oriented Knowledge Representation to Integrate Learning and Knowledge Elicitation in NeoDISCIPLE. *Knowledge Acquisition*, 6(1).
- Tecuci, G., Hieb M.R., Hille D. & Pullen J.M. (1994). Building Adaptive Autonomous Agents for Adversarial Domains, *Proceedings of the AAAI 94 Fall Symposium – Planning and Learning: On To Real Applications*.
- Tecuci, G. & Kodratoff Y. (1990). Apprenticeship learning in imperfect theory domains. In Y. Kodratoff & R. S. Michalski, Eds. *Machine Learning: An Artificial Intelligence Approach*. Vol. III, Morgan Kaufmann.
- White, E.L., Frosch K.E., Laviano, V.P., Hieb, M.R., Pullen, J.M. (1996). "Interfacing External Decision Processes to DIS Applications," *6th Conference on Computer Generated Forces and Behavioral Representation*.

10. Authors' Biographies

Michael Hieb received his PhD in Information Technology at George Mason University in 1996. Dr. Hieb is currently working for SAIC on the Multiple Reconfigurable C4I Interface (MRCI) project. He is developing VCDR, an instructable ModSAF agent, at the Center for Excellence in Command, Control, Communications and Intelligence at GMU. He has published over 20 papers in the areas of learning agents, knowledge acquisition and multistrategy learning. When working for IntelliTek, Dr. Hieb implemented a distributed problem-solving testbed at the Goddard Space Flight Center. Previously, he worked as a Nuclear Engineer for General Electric.

Gheorghe Tecuci is Professor of Computer Science at George Mason University. He has published over 70 scientific papers, mostly in the area of artificial intelligence. Gheorghe Tecuci is a member of the Romanian Academy and is known for his pioneering work on multistrategy machine learning and its integration with knowledge acquisition. He developed Disciple, which is one of the first multistrategy learning systems, and co-edited the first books on multistrategy learning and on the integration of machine learning and knowledge acquisition. He was the program chairman of the first workshops in these areas (MSL-91, MSL-93, IJCAI-93: ML & KA).

J. Mark Pullen is Associate Professor of Computer Science at George Mason University and a member of the Center for Excellence in Command, Control, Communications and Intelligence. Dr. Pullen was with the Defense Advanced Research Projects Agency (DARPA) from 1986 to 1992, where he was Program Manager for Advanced Computing, Networking and Distributed Simulation. His research interests include networking and distributed computing systems, and their applications to educational and military simulations.

Learning the Selection of Reactive Behaviors

Sumeet Rajput, Clark R. Karr, Jaime E. Cisneros, and Rebecca J. Parsons

Institute for Simulation and Training
3280 Progress Dr., Orlando, FL 32826
srajput@ist.ucf.edu

1. Abstract

Computer Generated Forces (CGF) systems are typically rule-based systems in one form or another. The behaviors of vehicles and units are implemented by a set of unchangeable rules. It may not be possible to consider all the nuances of a situation to develop a complete set. Research demonstrates a mechanism whereby the Subject Matter Expert (SME) can directly teach a CGF system a set of rules without an intermediary knowledge engineer (to develop production rules) or computer programmer (to write computer code). Machine learning is a research area in the Artificial Intelligence domain whose focus is on making machines "learn." In the CGF domain, machine learning can be used to teach simulated commanders new rules for responding to situations resulting in intelligent behavior selection; thus, "canned" responses are eliminated. A product of IST's research is a Learning Testbed which provides an environment for machine learning in the CGF domain. This testbed has been implemented in the Modular Semi Automated Forces (ModSAF) CGF system. The focus of this research is teaching ModSAF company commanders how to select appropriate reactive behaviors.

2. Introduction

2.1 Objective of this Research

CGF systems are typically rule-based systems. The behaviors of vehicles and units are implemented by a set of unchangeable rules. Although, rule designers spend significant effort in developing an "adequate" set of rules, the rule set is rarely, if ever, sufficient to address all possible situations. Rule-based systems suffer from suggesting similar responses to situations which may be significantly different.

Machine learning is a research area in Artificial Intelligence (AI) whose focus is on making machines "learn." After learning, the machine is expected to "solve" problems presented to it. In the CGF domain, machine learning can be used to teach simulated commanders new rules for responding to situations resulting in better or more intelligent behavior.

The primary objective of IST's research was to create a Learning Testbed. This testbed has been implemented in the ModSAF CGF system and has been used for "teaching" simulated company commanders how to choose a reactive behavior to a situation. The learning target was feasible and provided an opportunity whereby the existing reactive behavior mechanism could be improved. Further, the improved reactive behavior mechanism could be compared and contrasted with ModSAF's original reactive behavior mechanism.

3. Machine Learning

3.1 Supervised and Unsupervised Machine Learning

Learning is an important aspect of human cognition. Humans have the ability to acquire new knowledge, to learn new skills, and to improve with practice. One method to improve computer system performance is for the system to "learn;" i.e., acquire knowledge and change the performance based in a manner similar to human learning. Research into machine learning has revealed methods whereby computer systems can "learn," such as: instruction, analogy, examples, failure, observation, and discovery (Charniak et al. [1987]). These methods can be grouped into two disjoint categories: supervised and unsupervised machine learning methods (Hertz et al. [1992]).

In supervised machine learning, a teacher guides the student (i.e., the learning system) to a solution, or gives the solution to the student, along with an explanation. In unsupervised machine learning, students do not have a "teacher" or an "oracle" for guidance. Section 3.2 lists general machine learning approaches. These approaches are a mixture of supervised and unsupervised machine learning methods. In the remainder of the text the term "learning" will mean "machine learning" and the two terms will be used interchangeably.

3.2 General Machine Learning Methods

Learning methods are:

- *Connectionist*: Neural networks and their relatives (Adeli et al. [1995]) and (Hertz et al. [1992]).
- *Genetic/Evolutionary*: Genetic algorithms (Holland [1975] and Goldberg [1989]), classifier systems, and genetic programming (Adeli et al. [1995]), Koza [1992], and (Winston [1992]).
- *Inductive methods*: Decision tree systems and learning by example (Winston [1992]).
- *State Operator And Result (SOAR) Chunking* (Winston [1992]) and (Michalski et al. [1994]).
- *Case-based and other analytical methods*: Explanation-based learning (Winston [1992]), case-based learning (Kolodner [1993]), and exemplar-based learning (Bareiss [1989]).

Rajput and Karr [1995] presents a survey of the general learning methods and methods specific to learning reactive behaviors.

3.3 Learning Reactive Behaviors using Exemplar-based Learning

Exemplar-based learning is based on classification and problem solving (Bareiss [1989]). It combines learning with problem solving to learn new concepts and to refine existing concepts based on experience. From a simple classification point of view, common concepts are collected into categories. However, the basis for category membership of a concept is poorly understood. Creating a category, and classifying new cases as members of that category, depends on determining commonalities between new cases and existing members of the category.

Exemplar-based learning represents a category by a set of retained cases, called *exemplars*. Every exemplar in a category has an explanation associated with it to explain its degree of relevancy in that category. To classify a new case, an exemplar is retrieved to serve as a model for interpreting features of the new case. The model exemplar provides information as to which features the new case should

possess and their importance to the category's membership. Because there are several exemplars defining a category, a wide range of models is available to help classify typical as well as atypical cases that belong a given category.

For acquiring knowledge about when to use reactive behaviors, a classification hierarchy was created where each category represents a ModSAF reactive behavior. No problem solving is required to classify the reactive behavior exemplars because the SMEs classify the reactive behavior for a given situation.

4. The Learning Architecture

4.1 Introduction

Traditional AI has generally interpreted the organized nature of everyday activity in terms of plans and plan-following (Agre [1988]). Chapman [1987] has shown planning to be computationally intractable in all but simple descriptions. This poses a severe restriction for real time simulation. Because of the situated and interactive nature of units in the simulation environment, traditional planning is unsuitable to the realtime selection of reactive behaviors..

Agre [1988] takes a different approach in his computational theory of action. The principal idea is that continually redeciding what to do is more flexible and computationally feasible than executing a plan because it is more responsive to opportunities and contingencies. It is possible to approximate the ideal of continual redecision because life is almost entirely routine. The routine portion of the reasoning leading to each moment's action can be implemented efficiently by recording the reasons behind any novel bits of reasoning (Agre [1988]).

For learning reactive behaviors, the situation assessment provides the novel bits of reasoning to help a unit decide what to do. The situation assessment serves as an index to the appropriate reactive behavior stored as exemplars (Section 3.3). The bit patterns describing the situation are independent, so a rule will not invalidate another rule unless there is an attempt to map one situation to two different reactions. In this case the SME is asked to choose between two contending rules or to better qualify the situation, to prevent inconsistencies in the acquired knowledge.

4.2 Overview of the Approach

Machine learning begins with the detection of events that trigger reactive behaviors, such as presence of enemy ground and air units, minefields, and indirect fire. The company commander then analyzes the situation and consults a Knowledge Base (KB) for a reactive behavior to apply to the situation. If the KB cannot provide an answer, an SME is consulted. The SME selects a reactive behavior and then justifies his or her choice by selecting one or more pre-conditions or "justifications" through an editor. These justifications are considered:

1. Availability of cover and concealment in the situation.
2. Ratio of unit strength to enemy strength at the objective.
3. Ratio of total friendly strength to enemy strength at the objective.
4. Ratio of unit strength to total enemy strength.
5. Ratio of total friendly strength to total enemy strength.
6. Presence of dangerous threat.
7. Presence of friendly support.
8. Ratio of enemy strength on left, front, right, and rear to unit strength.
9. Inadequacy of unit strength to the mission.
10. Enemy operational activity.
11. Direct and indirect fire,
12. Distance to the objective.
13. Attacking aircraft.
14. Presence of minefield.

The justifications form the bit pattern described in Section 4.1 and together with the reactive behavior constitute a *rule*. The rule is then stored in the KB and the systems "learns." Thus, at any time a company commander's knowledge consists of the set of rules in the KB.

4.3 The Learning Algorithm

The Learning Algorithm (Figure 1) is implemented as a ModSAF task or Finite State Machine (FSM) running on behalf of a unit commander.

The algorithm begins in the "Monitor" stage. In this stage the algorithm checks for events that trigger reactive behaviors: establishment of Line-of-Sight to an enemy, air attack, detection of minefield, and indirect fire. After an event is detected, the algorithm goes to "Situation Analysis" where the unit commander does situation assessment (Section 4.4).

The algorithm checks the KB (Section 4.5.4) for a rule that matches the situation. If a rule is found it is presented to the SMEs. The SMEs have two choices: they can either accept the rule, in which case the reaction associated with the rule is executed, or they can modify it. If the SME modifies a matched rule the system "learns." The newly created rule is merged into the KB. This entails creating a new rule and possibly modifying the matched rule. In any case, after SMEs have provided their input, the chosen reaction is executed. If no rule is found that matches the situation, SMEs are asked to create one. This new rule is stored in the KB and its associated reaction is executed. After the reaction is over the algorithm returns to the "Monitor" stage.

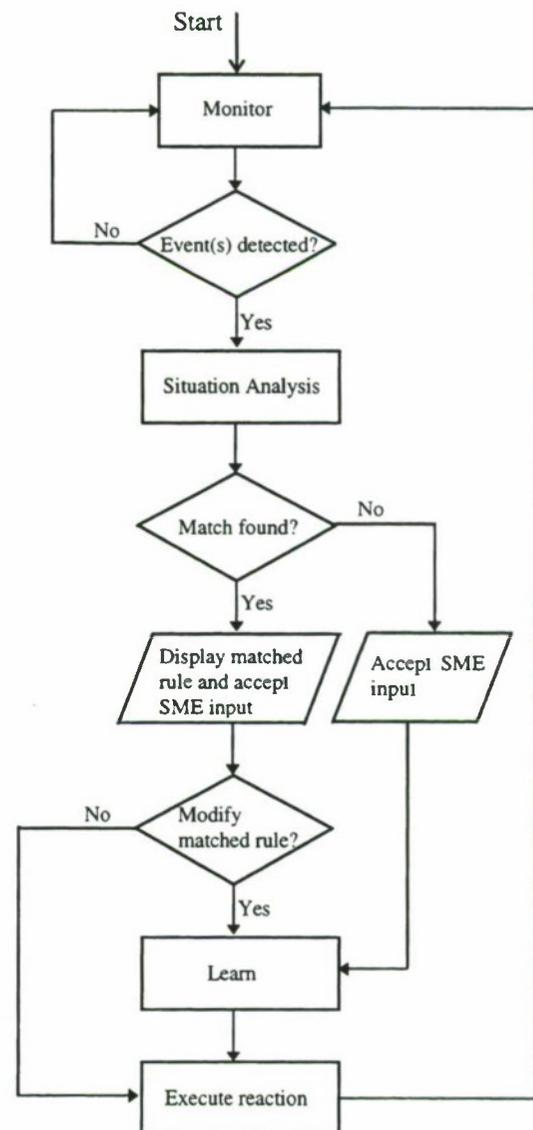


Figure 1: The Learning Algorithm.

4.4 The Situation Analyzer

Situation Analysis (or Situation Assessment) quantifies the situation with respect to certain parameters discussed in the following sections. Using the quantified situation assessment parameters, the system matches a rule with the situation. The Situation Analyzer (SA) is a software module developed for this purpose. It provides the simulated company commander with information about:

- itself (self assessment),
- enemies (enemy assessment), and
- the terrain (terrain assessment).

4.4.1 Self Assessment

This includes information about the unit and other friendly forces:

1. *Operational activity*: This is the unit's operational activity including halt, march, assembly area, hasty occupy position, attack, and others.
2. *Original strength*: Strength of the unit when it was created. Each vehicle is assigned a strength value representing its strength relative to another vehicle (Rajput and Karr [1995]). The unit's strength is the sum of vehicles' strengths in the unit.
3. *Current strength*: This equals original strength less the strength of vehicles damaged or destroyed.
4. *Type*: Type of unit including armor, artillery, mechanized infantry, and others.
5. *Speed*: The average speed of all vehicles in the unit.
6. *Receiving Indirect Fire (IF)*: Is the unit under indirect fire?
7. *Presence of friendly forces*: Are there visible friendly forces?
8. *Friendly forces strength*: If there are visible friendly forces, their strength is computed. Two values are computed: the average friendly forces strength and the strength of the strongest friendly group.
9. *Importance to mission*: Whether the unit is a main, support, or diversionary unit.
10. *Distance to objective*.
11. *Attacking aircraft*: Is the unit under attack from enemy aircraft?
12. *Receiving Direct Fire (DF)*: Has the unit received direct fire from enemy units within

a set time interval prior to the determination?

13. *Minefield detected*: Has the unit encountered a minefield?

4.4.2 Enemy Assessment

Visible enemy vehicles are separated into groups using a technique discussed in Cisneros et al. [1995]. Then, for each enemy group this information is deduced:

1. *Enemy strength*: Sum of the strengths of the vehicles in the group.
2. *Position*: The position of the group, whether left, front, right, or rear, with respect to the center of mass and heading of the analyzing unit.
3. *Distance*: Distance to the analyzing unit.
4. *Type*: The type of the group is the type of the vehicles that are similar and a majority in the group. For example, if the majority of vehicles in a group are T80s (an armored vehicle), the group is classified as an armor unit.
5. *Speed*: The average speed of the vehicles in the group.
6. *Operational activity*: Formation, speed, and vehicle headings are used to give clues to the enemy's operational activity. Vehicles in a platoon exhibiting high variance in heading are in defense. Vehicles in a platoon moving in line formations are assaulting whereas vehicles moving in other formations are traveling. Vehicles in a platoon that are not moving and are not in a defensive posture are holding.
7. *Enemy at or near the objective*: Is the enemy at or within a threshold distance from the objective?

4.4.3 Terrain Assessment

Terrain assessment computes covered and concealed positions within an area using available ModSAF routines.

4.5 Knowledge Representation

The system's knowledge is stored in a data structure called a KB. The fundamental unit of knowledge is a rule containing a bit pattern and a reactive behavior. The KB is used for:

- matching a rule's bit pattern to a situation,
- storing new rules, and
- modifying matched rules.

4.5.1 Situation Pattern, Justifications, and Rules

The SME "teaches" the simulated company commander which reactive behavior to execute in response to a situation. After SMEs have selected a reactive behavior, they justify their choice by selecting one or more *justifications*. The set of justifications encode a *bit pattern* which indicates why a reactive behavior was selected and details the necessary preconditions for the execution of a reactive behavior. Some justifications are more important than others and a measure of their importance is given by a *degree of importance*. This is an integer value, in the range 1 to 10, 1 being the least important and 10 being the most important. In the Learning Testbed, all justifications have the same degree of importance, namely 10. However, the system provides support for replacing the default value by one determined by the SME.

In many AI systems, rules are predefined and unchangeable. A rule's "if" part must be satisfied before the "then" part is executed (Winston [1992]). In IST's approach, rules can be modified during run time and new rules can be created from existing ones. As the system's repertoire of rules grows, so does its knowledge.

There are three types of justifications:

- *Binary*: A binary justification becomes part of a bit pattern when selected by the SME, otherwise it does not. For example, when the SME selects the justification, "Availability of cover and concealment in the situation," it means that the SME considers the availability of cover and concealment a precondition for selecting the reactive behavior. No other data is stored with the justification.
- *Enumeration*: The justification consists of a set of values. If any value is chosen, the justification becomes part of the bit pattern and the value is stored with the bit pattern. For example, the justification, "Enemy Operational Activities," has values "Holding," "Move," "Assault," and "Hasty Defense." The SME picks one value, such

as Move, from the set which is stored with the bit pattern.

- *Range*: Two numbers are associated with the justification which define the lower and upper bounds of a range. The justification becomes part of the bit pattern when selected by the SME and the range bounds are determined from situation variables (Section 0).

Based on consultations with IST's SME, these justifications were considered:

1. *Availability of cover and concealment in the situation (Binary)*: Important if selected.
2. *Ratio of unit strength to OPFOR strength at objective Favorable (Range)*:

$$\text{ratio} = \frac{S_u}{S_{OPFOR(obj)}}$$

where S_u is the strength of the analyzing unit and $S_{OPFOR(obj)}$ is the OPFOR's strength at the objective.

3. *Ratio of unit strength to OPFOR strength at objective unfavorable (Range)*:

$$\text{ratio} = \frac{S_{OPFOR(obj)}}{S_u}$$

4. *Ratio of total friendly strength to OPFOR strength at objective favorable (Range)*:

$$\text{ratio} = \frac{S_f}{S_{OPFOR(obj)}}$$

where S_f is the total friendly strength.

5. *Ratio of total friendly strength to OPFOR strength at objective unfavorable (Range)*:

$$\text{ratio} = \frac{S_{OPFOR(obj)}}{S_f}$$

6. *Ratio of unit strength to total OPFOR strength favorable (Range)*:

$$\text{ratio} = \frac{S_u}{S_{OPFOR}}$$

where S_{OPFOR} is the total OPFOR strength.

7. *Ratio of unit strength to total OPFOR strength unfavorable (Range):*

$$\text{ratio} = \frac{S_{OPFOR}}{S_u}$$

8. *Ratio of total friendly strength to total OPFOR strength favorable (Range):*

$$\text{ratio} = \frac{S_f}{S_{OPFOR}}$$

9. *Ratio of total friendly strength to total OPFOR strength unfavorable (Range):*

$$\text{ratio} = \frac{S_{OPFOR}}{S_f}$$

10. *Presence of dangerous threat (Range):* Two ranges are stored: The average strength and the average distance to all enemies.

11. *Presence of friendly support (Range):* Two ranges are stored: The average strength and the average distance to all friendly forces.

12. *Enemy on right flank (Range):* Ratio of OPFOR strength on the right flank to unit strength.

$$\text{ratio} = \frac{S_{OPFOR(r)}}{S_u}$$

where $S_{OPFOR(r)}$ is the OPFOR strength on the right flank.

13. *Enemy on left flank (Range):* Ratio of OPFOR strength on left flank to unit strength.

$$\text{ratio} = \frac{S_{OPFOR(l)}}{S_u}$$

where $S_{OPFOR(l)}$ is the OPFOR strength on the left flank.

14. *Enemy in the front (Range):* Ratio of OPFOR strength in the front to unit strength.

$$\text{ratio} = \frac{S_{OPFOR(f)}}{S_u}$$

where $S_{OPFOR(f)}$ is the OPFOR strength in the front.

15. *Enemy in the rear (Range):* Ratio of enemy strength in the rear to unit strength.

$$\text{ratio} = \frac{S_{OPFOR(rear)}}{S_u}$$

where $S_{OPFOR(rear)}$ is the OPFOR strength in the rear.

16. *Force inadequate due to losses (Range):* Ratio of current unit strength to original unit strength.

$$\text{ratio} = S_u / S_{original}$$

where $S_{original}$ is the original unit strength.

17. *Enemy operational activity (Enumeration):* The enemy operational activity chosen by the SME is stored in the justification.

18. *Direct fire (Binary):* Important if selected.

19. *Indirect fire (Binary):* Important if selected.

20. *Far from objective (Range):* Distance to the objective is stored as a lower bound in the justification.

21. *Close to objective (Range):* Distance to the objective is stored as an upper bound in the justification.

22. *Attacking aircraft (Binary):* Important if selected.

23. *Minefield detected (Binary):* Important if selected.

4.5.2 Matching Rules To A Situation

After rules have been created and stored, they have to be searched to find one that matches the situation. Two types of matches are determined: perfect matches and near matches.

A rule is a **perfect** match if the situation satisfies all the justifications. A binary justification is satisfied if the situation has the same property as the justification. For example, if the justification *Cover and Concealment* is selected and the situation also has cover and concealment, the justification is satisfied. An enumeration justification is satisfied if the enumeration value in the justification is present in the situation. For example, if the justification *Enemies Operational Activity* has the value "move" and the enemy in the situation is also moving, the justification is satisfied. A range justification is satisfied if the situation variable falls within the specified range. For example, if the justification *Unit to OPFOR* has the range 5.09 to ∞ and the ratio of unit strength to OPFOR strength in the situation is 10.0:1.0, the justification is satisfied because 10.0 falls within the range 5.09 to ∞ . On the other hand, a rule is a **near** match if some justifications are true and the sum of the degree of importance (score) of true justifications exceeds a threshold.

Of the perfect and near matched rules the best-fit rule is returned to the SME. This could either be a perfect match rule or a near match with the highest score.

4.5.3 Creating Rules and Modifying the Best-fit Rule

If the system cannot find a rule that matches the situation, the SME can create and insert a new rule in the KB. On the other hand, if the system determines a match and presents the best-fit rule to the SME, the SME can modify it. There are three cases:

- *SME chooses new justifications (Case 1):* This results in a new rule which is inserted in the KB. The best-fit rule remains unchanged.
- *The ranges change (Case 2):* A new rule is inserted into the KB. The new rule has the same justifications as the best-fit rule but differs in the ranges of those justifications. The best-fit rule is modified.
- *Only the reaction changes (Case 3):* The SME selects a new reaction to the situation for which a different reaction was chosen previously. The reaction in the best-fit rule is overwritten by the SME's selection.

Consider Case 2 which requires that the best-fit rule be modified. The modification is in terms of eliminating range overlap occurring as a result of

creating a new rule from a best-fit rule. Overlapping ranges may lead to multiple matched rules in a future situation, the matched rules differing only in their range bounds. This leads to ambiguous results. For example, assume two rules, R_i and R_j . R_i is the best-fit rule and R_j is derived from it. The only difference between R_i and R_j is in the difference in their range bounds of a justification x . Let r_i and r_j be the ranges of x in R_i and R_j respectively.

Now, if the corresponding situation variable has value v which lies in both r_i and r_j , rules R_i and R_j will be perfect matches. The solution is to split the ranges as shown by this example:

Let $r_i = [10.0, \infty]$, $r_j = [20.0, \infty]$, and $v = 25.0$; i.e., v lies in both r_i and r_j . The ranges are split such that $r_i = [10.0, 20.0]$ while r_j remains unchanged. Now, the ranges do not overlap and ambiguous situations are avoided.

4.5.4 The Knowledge Base Organization

To match rules efficiently, the KB is organized hierarchically (Figure 2).

The rules are organized for indexing and retrieval with respect to unit type; for example, $UNIT_TYPE_1$ might be an armor company and $UNIT_TYPE_2$ might be a mechanized infantry company.

Within each unit type the rules are further organized based on the unit's operational activity (oa); for example, oa_1 might be a "march" and oa_2 might be an "assault." Each operational activity node has a list of rules; for example, $rules_{1..n}$ in the figure. The broken arrows represent the continuation of the KB for other unit types. This organization is in addition to the exemplar-based hierarchy discussed in Section 3.3.2.

In a given situation, only the branch of the hierarchy containing the unit's type and operational activity is searched. This avoids searching the entire KB and constrains the search to a narrow area.

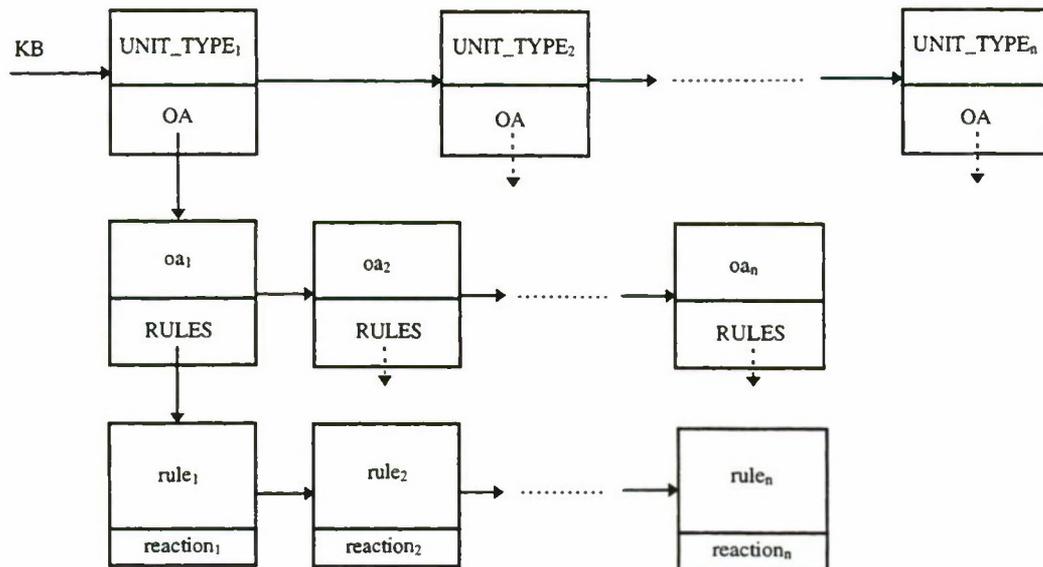


Figure 2: The Knowledge Base.

5. Results

The learning testbed was implemented in ModSAF version 1.5.1 and used for "teaching" company commanders the reactive behaviors for different situations. To test the learning testbed, IST devised an experiment. The experiment was implemented in two parts: training the company commanders and using the learned reactive behaviors.

In the first phase of the experiment, four sets of scenarios were developed that resembled typical battlefield conditions. These sets comprise the training set (Section 5.1). Using these sets of scenarios, IST's SME trained the company commanders with different reactive behaviors for different situations. This information was input to the KB (Section 4.5.4).

In the second phase of the experiment, the KB created for each scenario set was used in each of the other three scenario sets. The applicability of the reactive behavior, recommended by the KB, to the situation was judged by IST's SME (Section 5.2).

5.1 The Training Set

IST developed four sets of scenarios for the training set (Rajput and Karr [1995]). Each set had a base scenario which was varied to develop the entire set. The variations resulted in a richer training set allowing more justifications to be used. The four scenarios were executed on different machines and

four KBs were created and saved as four output files. Table 1 shows the justifications used by all scenarios.

Justification	A	B	C	D
Cover and Concealment	√	√	√	√
$S_u/S_{OPFOR(obj)}$				
$S_f/S_{OPFOR(obj)}$				
S_u/S_{OPFOR}	√	√	√	√
S_f/S_{OPFOR}	√	√	√	√
Dangerous threat	√	√		√
Friendly Support				
$S_{OPFOR(l, r, f, rear)}/S_u$		√	√	√
$S_u/S_{original}$				
Enemy operational activity			√	√
Direct or indirect fire		√		
Distance to the objective				
Attacking aircraft				
Presence of minefield		√		

Table 1: Justifications used in the scenarios.

5.2 Using Learned Reactive Behaviors

After the scenarios were executed, four KBs were created. To test the learned knowledge, each KB was applied to all scenarios which did not originally create it; a KB, KB_i , was applied to all scenarios, S_j , such that $i \neq j$. For IST's experiment, this yielded 12 combinations. The performance of KB_i in a scenario S_j was judged subjectively by IST's SME. The SME felt that the recommended reactive behaviors were

correct in 75% of the cases within the constraints of the ModSAF system.

6. Conclusions

In traditional CGF systems, such as ModSAF, two situations, which may require different responses, elicit the same reactive behavior from a company. Improving the choice of a reactive behavior provided an interesting topic for research. This project implemented a learning testbed that was successfully used to learn reactive behaviors.

IST's approach uses supervised learning (exemplar-based learning), allowing a company commander the ability to learn how to react to different situations. An SME decides which reactive behavior to use under certain conditions. This information is stored in a KB, in the form of a bit pattern and semantic information. (This is referred to as a "rule.") Future simulation situations may yield a rule that matches the situation, in which case the rule's reactive behavior is used as a response to the situation.

7. References

- Adeli, Hojjat and Hung, Shih-Lin (1995). Machine Learning - Neural Networks, Genetic Algorithms, and Fuzzy Systems, John Wiley & Sons, 1995.
- Agre P (1988). The Dynamic Structures of Everyday Life, Ph.D. dissertation AI-TR 1085, Massachusetts Institute of Technology, October 12, 1988, Cambridge MA.
- Bareiss, R (1989). Exemplar-Based Knowledge Acquisition: A Unified Approach to Concept Representation, Classification, and Learning, Academic Press, Inc., San Diego, CA.
- Chapman, D. (1987). "Planning for Conjunctive Goals," *Artificial Intelligence*, 32(3), pages 333-378.
- Charniak, E., and McDermott, D. (1987). Introduction to Artificial Intelligence, Addison Wesley Publishing Company, Inc., Reading, MA.
- Cisneros, J. E., Karr, C. R., and McCauley-Bell, P. (1995). "Intelligent Targeting in ModSAF," *Contract Report IST-CR-95-36*, Institute for Simulation and Training, University of Central Florida.
- Goldberg, David E. (1989). Genetic Algorithms in Search, Optimization, and Machine Learning, Addison Wesley Publishing Company.
- Hertz, J., Krogh, A., and Palmer, R. G. (1992). Introduction To The Theory of Neural Computation, Addison Wesley Publishing Company, Inc., Redwood City, CA.

Holland, John H. (1975). Adaptation in Natural and Artificial Systems, The University of Michigan Press, Ann Arbor, MI.

Kolodner, J. (1993). Cased-Based Reasoning, Morgan Kaufmann, 1993.

Koza, J. R. (1992). Genetic Programming: On the Programming of Computers by Means of Natural Selection, MIT Press.

Koza, J. R. (1994). Genetic Programming II: Automatic Discovery of Reusable Programs, MIT Press.

Michalski, R. and Tecuci, Gheorghe (1994). Machine Learning - A Multistrategy Approach, Volume IV, Morgan Kaufmann Publishers, San Francisco, California.

Winston, P. H. (1992). Artificial Intelligence, Third Edition, Addison-Wesley Publishing Company, Inc., Redwood City, CA.

8. Acknowledgment

This research was sponsored by the US Army Simulation, Training, and Instrumentation Command as part of the Intelligent Simulated Forces project, contract N61339-92-C-0045. That support is gratefully acknowledged.

9. Authors' biographies

Sumeet Rajput is an Associate Computer Scientist in the Intelligent Simulated Forces project at the Institute for Simulation and Training. Mr. Rajput has a Master of Science degree in Computer Science from the University of Central Florida and is an MBA student at the University of Central Florida. His research interests are in the areas of Computational Geometry, Physical Modeling, and Computer Generated Forces.

Clark R. Karr is a Program Manager and the Principal Investigator of the Intelligent Simulated Forces project at the Institute for Simulation and Training. Mr. Karr has a Master of Science degree in Computer Science. His research interests are in the areas of Artificial Intelligence and Computer Generated Forces.

Jaime E. Cisneros was an Associate Computer Scientist in the Intelligent Simulated Forces project at the Institute for Simulation and Training. Mr. Cisneros has a Master of Science degree in Computer Science from the University of Central Florida. His research interests are in the areas of Natural

Language Understanding, Machine Learning, and Computer Generated Forces.

Rebecca J. Parsons is an Assistant Professor of Computer Science at the University of Central Florida. Dr. Parsons received her Bachelors of Science in Computer Science and in Economics from Bradley University and her Masters of Science and Doctor of Philosophy in Computer Science from Rice University. Her research interests are Programming Language Semantics applied to Distributed and Parallel Computation, Computataional Biology, Genetic Algorithms and Machine Learning.

An Intelligently Interactive Non-Rule-Based Computer Generated Force

Lawrence J. Fogel, V. William Porto, and Mark Owen
Natural Selection, Inc.
3333 N. Torrey Pines Court, Suite 200
La Jolla, CA 92037
(619) 455-6449
(619) 455-1560 fax

1. Abstract

Attempts to optimize behaviors are often realized through the imposition of heuristics in an artificial intelligence program. A set of if-then-else rules are derived and applied to the problem at hand. This approach, while mimicking the previously discovered decisions of humans, does not really allow for true, dynamic learning. In this paper we discuss the use of evolutionary programming to optimize computer generated forces (CGF) behaviors which actually learn courses of action adaptively, as opposed to relying on a preset rule base. Actions of computer generated forces are created on-the-fly by iterative evolution through the state space topography. Possible courses of action at each time step in the scenario are scored with respect to a payoff matrix (Valuated State Space) which is goal specific. This methodology is inherently self-adaptive to the dynamic environment of the CGF.

2. Introduction

Effective training requires realistic simulation of the combat environment including the enemy force, its decision-making ability, and mission. Having qualified individuals simulate the OPFOR command presumes we understand their doctrine and can adequately reflect their culture. Both these assumptions are false. In addition, such simulations are non-repeatable and cannot be calibrated.

It is equally dangerous to train against an enemy that follows any set of rules derived from prior combat experience. Particular rules may have been effective at that time in that setting, but it is dangerous to re-fight the old war. Small changes in behavior may have a major effect on the outcome. Tomorrow's enemy may be more intelligent and have a new mission.

Indeed, training against an enemy that follows of *any* set of fixed rules is inappropriate, for the real enemy learns, may demonstrate initiative, and thus behave in a generally unpredictable manner. The computer generated force must be adaptive for, in the words of Charles Darwin, "It is not the strongest of species that survive . . . but rather the one most responsive to change." Facing an expert system, we learn how to defeat the game rather than an intelligently interactive foe.

What is needed is an arbitrary-culture, intelligently-interactive computer-generated adversary that can operate at any specified level of intelligence from, say, inept to ingenious. It must be able to take full advantage of the available sensors, communication/computation capabilities, and weapons/platforms or, for the sake of planning future missions, those of an other time and place. This capability can be realized through the use of the Valuated State Space (VSS) approach and evolutionary programming. The former provides a convenient way to express the enemy's mission in measurable terms. The latter discovers increasingly appropriate courses of action in light of that mission until one of sufficient worth is found, or the assigned computation has been expended.

2.1 Discussion

Proper assignment of forces begins with a clear understanding of what must be achieved, by when. But what if that outcome is not realized? Surely, some value is found in lesser degrees of achievement. There are even times when our primary concern is to avoid some particularly undesirable outcome. In other words, to be meaningful, the mission must be stated in terms of the significantly different futures and their relative worth, all the way from utopia to catastrophe, for only then can we measure the overall

worth of any situation and properly identify the remaining problems.

Unfortunately, it is difficult to envision these significantly different futures, no less their relative worth. The Valuated State Space (VSS) approach provides a way to overcome this difficulty. Those responsible for defining the mission indicate preferentially independent aspects of their concern. Each of these parameters is weighted in relative importance and made measurable in terms of those differences that make a difference in degree of achievement. Each of these defined class intervals is then attributed some value. Thus each line item is a multiple-choice question concerning the current or any projected situation. A normalizing function that expresses the relationship among the parameters translates the answers to the individual questions into the overall worth of that situation. In many situations it is appropriate to use the weighted arithmetic mean. If, however, all the parameters are critical the weighted geometric mean is appropriate, and there are various degrees of criticality. Clearly, Measures of Effectiveness (MOEs) and Measures of Performance (MOPs) alone do not tell the whole story.

In practice, the mission takes the form of a hierarchy of measurable parameters and subparameters together with an appropriate normalizing function across the various levels. Briefly stated, the Valuated State Space and normalizing function indicate what to measure, with what specificity, and how to fuse these data into the overall worth of any situation. It identifies the remaining deficiencies/problems by priority, as well as the overall worth of any prospective solution.

But our best supposed move may not truly be best if it significantly injures an ally and/or greatly benefits a foe. The Valuated State Space approach can be expanded to include the presumed purpose of each of the other players. We can then find our best move (and, if we choose, their best moves) in light of the mutual attitudes and the current state of the game. With the mission well defined it becomes appropriate to evaluate prospective "what if's," Courses of Action (COAs), tactics, for these are simply alternative temporal commitment of the allocable resources, combinations of the available personnel and equipment, modes of deployment, within the related dynamics, constraints, and doctrine. But the number of possible tactics is immense, a number so great as to forbid exhaustive analysis. The number of

those considered by the assigned personnel is, in comparison, minute. It is reasonable to believe that there are much better ways to accomplish the mission than any of those "on the table." What is needed is a way to efficiently search the space of possible tactics to find one of sufficient value in time for it to be useful.

When exhaustive analysis is clearly impossible, we ordinarily turn to heuristics. But these prove useful only under certain circumstance. For example, steepest descent is prone to failure if there are a multitude of minima points. Linear programming is often used even when the constraints are known to be nonlinear. Complex problems are decomposed into simple ones so that these can be treated separately. But the aggregate of these local optimizations leads to a global optimum only if the component problems are independent, and they rarely are. Statistical procedures generally presume stationarity, but the real-world is nonstationary. In fact, these and other heuristics are rules. If the rules that always solve the problem at hand are known a priori, the optimal approach is to use them. If they are not known, it is dangerous to guess, for the rules chosen may often stand in the way of finding a better solution.

In contrast, the evolutionary programming algorithm (Fogel et al., 1962, Fogel 1995) is a most general optimization technique. The *only* "rules" are problem-independent iterative mutation and selection. Those components of randomness which are found to be of value are retained to benefit in further generations of solutions. Evolutionary programming is an inherently elegant and potent technique simulating the mechanisms of natural evolution and selection to generate organisms which exhibit optimal behavior with regard to an environment and desired payoff function.

Evolutionary programming operates by iteratively generating successive populations of finite state machine organisms. A population of "parent" machines is exposed to the observed environment and measured with respect to their ability to predict the next event (e.g., course of action) in light of a prescribed payoff function.

Offspring machines are created by randomly mutating each parent machine. For convenience, each parent is often made to produce a single offspring, but generating multiple offspring per parent is also possible. Mutations are chosen with respect to a probability distribution, typically uniform. The

number of mutations per offspring is also chosen with respect to a probability distribution or it may be fixed *a priori*. These offspring are then evaluated over the existing environment in the same manner as their parents.

It is interesting to note that, in direct contrast, advocates of genetic algorithms adopt exactly the opposite position. They traditionally presume it's "a good idea" to code every problem into a string of bits (simulating chromosomes).

Genetic algorithms construct solutions bottom-up. Crossover operators are used to exchange hopefully useful building blocks of subcode between candidate

solutions. In contrast, evolutionary programming discovers solutions top-down. It only scores entire individuals in terms of their expressed behavior. Evolutionary programming maintains the interrelationships between the sections of subcode and the manner in which they fit together as a whole.

Those machines that provide a sufficient payoff are retained to become parents of the next generation. Typically, half of the total machines are saved so that the parent population remains the same. This process is iterated until it is required to make an actual prediction, i.e., create a plan of action for the next time step. The "best" machine is chosen to generate this prediction. Figure 1 diagrams this process pictorially.

Evolutionary Programming Algorithm

```

t:=0;
initialize  $P(0) := \{a'_{1(0)}, a'_{2(0)}, \dots, a'_{\mu(0)}\}$ 
evaluate  $P(0) : \{\Phi(a'_{1(0)}), \Phi(a'_{2(0)}), \dots, \Phi(a'_{\mu(0)})\}$ 
iterate
{
  mutate:  $P'(t) := m_{\Theta_m}(P(t))$ 
  evaluate:  $P'(t) : \{\Phi(a'_{1(t)}), \Phi(a'_{2(t)}), \dots, \Phi(a'_{\lambda(t)})\}$ 
  select:  $P(t+1) := s_{\Theta_s}(P'(t) \cup Q)$ 
  t := t + 1;
}

```

where

a' is an individual member in the population

$\mu \geq 1$ is the size of the parent population

$\lambda \geq 1$ is the size of the offspring population

$P(t) := \{a'_{1(t)}, a'_{2(t)}, \dots, a'_{\mu(t)}\}$ is the population at time t

$\Phi: I \rightarrow \mathfrak{R}$ is the fitness mapping

m_{Θ_m} is the mutation operator with controlling parameters Θ_m

s_{Θ_s} is the selection operator $\exists s_{\Theta_s} (I^\lambda \cup I^{\mu+\lambda}) \rightarrow I^\mu$

$Q \in \{\emptyset, P(t)\}$ is a set of individuals additionally accounted for in the selection step, i.e. parent solutions.

Figure 1: The evolutionary programming paradigm

Note that evolution is most properly simulated at the phenotypic rather than genotypic level, for natural selection acts only on expressed behavior, not on the individual organs or the genes. In the words of the famous biologist Ernst Mayr (1988), "The genes are not the units of evolution nor are they, as such, the targets of natural selection."

Genetic algorithms (Holland, 1975) are suitable when a problem can be successfully partitioned into subproblems that can be dealt with independently. Unfortunately, most real-world problems encountered rarely exhibit such simplicity. Evolutionary programming, because it acts top-down, is particularly appropriate when a problem is not easily separable, and each potential subproblem is affected by the solutions to other subproblems. That evolutionary programming outperforms the genetic algorithm on such problems (and often by orders of magnitude) has been repeatedly demonstrated in the scientific literature.

3. Implementation

Application of evolutionary programming involves consideration of several key aspects of a problem. These include problem representation, data flow, parameterization, and generating a function for measuring the relative worth of solutions in the population.

For our application of evolutionary programming to generating intelligently interactive CGF entities, the decision was made to utilize as many existing low-level MODSAF finite state machine behaviors as possible. Thus we utilized a state space which created a parameterized task list for each member of the population. An evolutionary programming task was scheduled to run periodically using the MODSAF scheduler. Figure 2 shows the overall system procedure in a block diagram format. This task first executed a temporary 'freeze scenario' command in order to prevent data synchronization problems. Data flow to and from MODSAF was implemented in the following manner. First, the current state of the world (e.g. entities, positions, actions, status) was obtained through querying the database directly.

Next, these state parameters were used to create a set of parent plans for each entity in the population. Offspring plans of action were created through mutation of the current task(s) and parameters within

Interactive Evolution of Task Plans using MODSAF

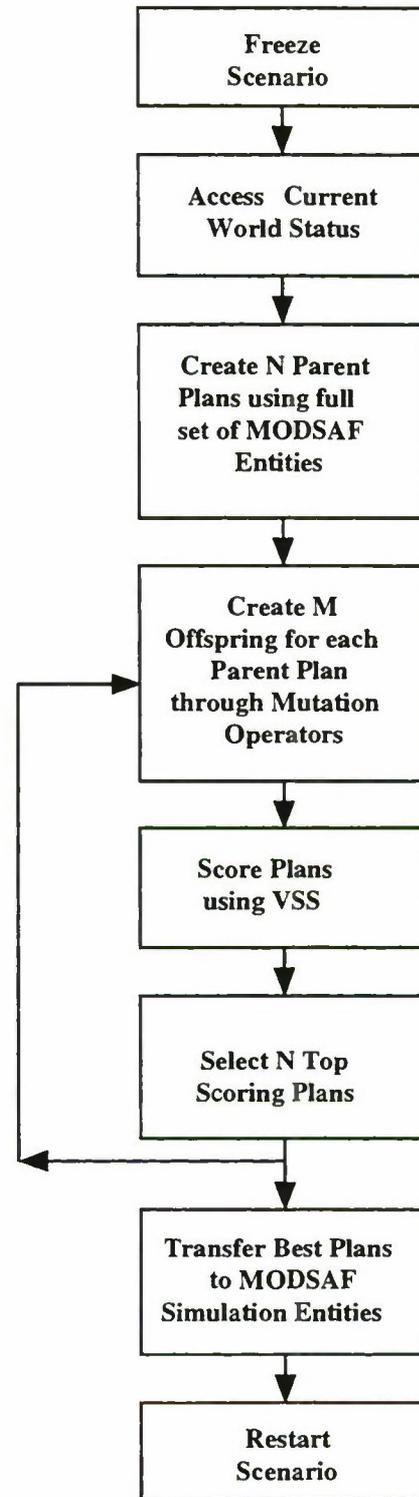


Figure 2: Block diagram of behavioral optimization through evolutionary programming.

the task(s). A copy of the parent plan was first made, and both the number and type of mutations were randomly chosen from a Poisson and uniform distribution, respectively.

The achievement of the specified goal by each plan in the population was measured by calculating the probabilities of being killed, killing opposing forces, and other pertinent features (attainment of positions, etc.). This scoring function is defined for each player in the scenario, and is not necessarily symmetric due to potential alliances and relative worths of achieving parts of a goal. Since execution of a task or course of action most often occurs over time, a kinematic state prediction mechanism was implemented. Current states were projected in discrete steps throughout the time of the evolved plan by using velocity and acceleration estimates. This allowed both cumulatively assessing the state through the discrete time steps, as well as allowing for multiple (temporally sequential) tasks.

After the desired number of iterations were processed, the final plans of action for each entity were taken from the 'best' (highest scoring) evolved population member. By implementing the process with this approach, we were able to realize not just single player (evolving side A versus a side B generated by a human or AI system) games, but also multiple player scenarios (evolved side A versus an evolved side B, C, ...). Other than by memory limitations, there is no inherent limit on the number of entities on each side nor of the number of sides playing against each other.

For multi-player games, a payoff function is generated which defines the specific goal (or purpose) of each side in the scenario. This methodology also allows for the inclusion of allied or even neutral sides.

Finally, after the iterative process is finished, the resulting plans for the desired number of sides are transferred back to MODSAF by affecting the current task state stack and parameterization of the new task(s). This is accomplished through modification of objects within the persistent object database. After all of the proposed changes have been performed, the scenario is 'unfrozen' so that it may perform normal MODSAF updates to the states until the next periodic call to the evolutionary program. No loss of generality is incurred by starting and stopping MODSAF for these optimization updates other than a slight degradation in visual performance of the

system. Typically, the program is executed on periodic 5 to 10 second intervals, and by governing the number of desired iterations, the optimization process produces very little apparent visual change in the MODSAF update rates. As faster computers are available, this will eventually be completely invisible to the operator. In fact, the optimization process can be implemented on a separate processor on an interrupt basis, with the only impact to MODSAF being the small time necessary to access and transfer parameters back and forth between computers.

All code was developed using the standard ANSI C processing language for maximum flexibility.

4. Experiments

A series of experiments were conducted to treat generic military situations using the combination of MODSAF and evolutionary programming for behavioral optimization. For the sake of simplicity, these initial experiments pitted two MODSAF entities against each other. The initial experiment concerns defensive movement. Here an entity (vehicle or platoon) is required to take minimum risk in moving to another location. For example, a single tank is required to run a gauntlet by moving along a road to reach the desired endpoint by a given time, taking minimum risk of being observed (attacked, damaged, or destroyed). The road is, say, ten miles long. The required time of transit is less than 30 minutes. The acceptable risk is less than one chance in 10 of being observed by the enemy.

More specifically, the purpose of the friendly tank is to complete the following mission:

6	Estimated time of transit
10	> 30 minutes
9	< 30 but > 40 minutes
6	< 40 but > 70 minutes
3	< 70 but > 110 minutes
1	< 110 but > 180 minutes
0	> 180 minutes, and
8	Probability of being observed
10	> 0.1
8	< 0.1 but > 0.3
5	< 0.3 but > 0.5
2	< 0.5 but > 0.7
1	< 0.7 but > 0.9
0	> 0.9

Note that, the relative important weights are arbitrary so that the mission can range from "transit without concern for risk" all the way to "only transit if there is no risk." Both parameters are critical so that the overall worth of any situation is the geometric mean of the contributions in each regard.

In this initial experiment the location of enemy observers is given, together with the likelihood of their observing the tank as a function of range. The required solution is the speed of the tank as a function of position during the transit.

A second experiment involves including the possibility of moving off road in the preceding scenario. This demonstrates the capability to generate plans which alter both the route and velocity of the entities. Additional experiments encompass the capability of attacking specified unit entities on the opposing side. Each unit entity in the scenario is given a priority of which the scoring function weighs outcomes. These priorities are not necessarily equal as this set of experiments is designed to test the capability for evolving behaviors which achieve are capable of attacking specific goals while minimizing risk encountered through the simulation.

Initial results of these tests have indicated the definite capability of the evolutionary program to generate interactively intelligent behaviors. The resultant paths and parameterizations thereof indicated learning of increasingly better and *adaptive* plans in light of the specified goals. Results of these experiments will be demonstrated at the conference as the behaviors are best presented graphically.

5. Future Directions

Future experiments will be conducted which stress increasingly complex goals. In addition, allocation of multiple units on a side will be tested, as well as two and three player games. Most of the improvements to the code will focus on implementing a more detailed state space (with additional MODSAF options) and subsequent mutations which can operate on these states.

Additional efforts will also focus on making the graphical interface more user friendly, with access to more parameters in the scenarios. A graphical presentation of the performance through time will also be developed. This will allow the user to view performance of the algorithm and make dynamic adjustments of the population size, mutation

strategies, and number of iterations per periodic update. Automatic adjustment of mutation parameters through self-adaptation will also be added to our program. This meta-level implementation of evolutionary programming holds great promise as the parameterization is self-adaptive, and leaves one less item of concern for the operator.

6. Acknowledgement

This work was funded by the Naval Air Warfare Center, Training Systems Division, Code N61339 under contract N61339-95-C-0088

7. References

- Fogel, D.B., 1995, *Evolutionary Computation*, Piscataway, NJ: IEEE Press.
- Fogel, L.J., Owens, A.J., and Walsh, M.J., 1966, *Artificial Intelligence through Simulated Evolution*, New York, NY: John Wiley.
- Holland, J.H., 1975, *Adaptation in Natural and Artificial Systems*, Ann Arbor, MI: University of Michigan Press.
- Mayr, E., 1988, *Toward a New Philosophy of Biology*, Harvard, MA, Belknap Press.

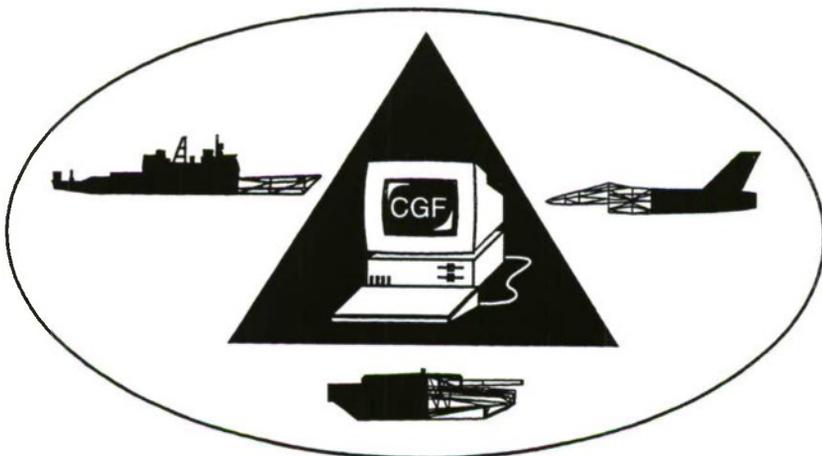
8. Authors' Biographies

Lawrence Fogel received his Ph.D in engineering from UCLA in 1964. Dr. Fogel is currently president of Natural Selection, Inc. and has pioneered much of the first reasearch in evolutionary programming in the early 1960s. He is co-author of *Artificial Intelligence Through Simulated Evolution*, published by John Wiley, 1966. Dr. Fogel holds six patents and has over 100 publications in journals, conferences, and edited volumes. His research interests include the engineering potential of evolutionary programming as well as the evolution of human intelligence and consciousness.

V. William Porto received his B.A. in mathematics and is currently completing his M.A. in applied mathematics from the University of California as San Diego. He is currently Vice President of Natural Selection, Inc., and is the program manager for the STRICOM project at NSI. Mr. Porto is on the editorial boards of the *IEEE Journal of Evolutionary Computation*, *IOP/Oxford Press Handbooks for Neural Computation and Evolutionary Computation*, and serves on the ONR advisory panel for neural networks. Current research interests include

optimization theory, neural networks, digital signal and image processing, and cryptography.

Mark Owen received his B.S. in electrical engineering at California State University at Long Beach, and is finishing his masters in E.E. He is currently a staff scientist with Natural Selection, Inc., and is active in the local chapter of the Control Systems Society of the IEEE. He has also worked previously in the areas of ocean surveillance and real-time control. Mr. Owen's interests include control systems, signal processing, neural networks and optimization theory.



Session 4b: Project Status Reports

Clarkson, NOSC NRaD

Naff, BDM Federal

Peacock, SAIC

Norwood, STRICOM, PM CATT



LeatherNet: A Synthetic Forces Tactical Training System For the USMC Commander

Jeffrey D. Clarkson
Naval Command Control and Ocean Surveillance Center, RDT&E Division (NRaD)
53570 Silvergate Avenue San Diego, CA 92152-5246
clarkson@nosc.mil

John Yi
KES
7790 Roan Road, San Diego, CA 92129
jyi@nosc.mil

1. Abstract

This paper presents a progress report on the LeatherNet Project and includes a description of the LeatherNet concept of operation, development concepts & approach, and system description & products to date. LeatherNet provides the interface for United States Marine Corps (USMC) commanders into the environment of the Defense Research Projects Agency (DARPA) Synthetic Theater of War '97 (STOW-97) exercise. Capabilities of LeatherNet facilitate the training of Marine commanders in tactical battlefield management techniques.

Computer tools currently implemented in LeatherNet include: Marine Corps Synthetic Forces (MCSF), Terrain Evaluation Module (TEM), CommandVu and CommandTalk Human Computer Interfaces (HCI). MCSF is developing the entities necessary to conduct Marine Corps operation in a Virtual Joint Task Force with emphasis on the development of individual combatants. TEM is a tool for terrain evaluation including weapons fans, line-of-sight analysis, and so forth. CommandVu provides an enhanced synthetic environment to display Marine Corps command decision tools and MCSF behaviors. CommandTalk is a natural language, speech recognition and gesture system that provides users with the ability to communicate to CommandVu and MCSF simulation through natural means. These tools when used together provide the means to conduct enhanced tactics preparation, simulation, rehearsal and after action review for the USMC.

2. Introduction

In the Fall of 1993 representatives from the Defense Advanced Research Projects Agency (DARPA) met

with the Commanding General of the Marine Corps Air Ground Combat Center (MCAGCC), Twentynine Palms, CA and reached a cooperative set of agreements. As part of the Synthetic Theater of War (STOW) program, DARPA would invest hardware and software resources at MCAGCC in order to develop the amphibious component of a virtual Joint Task Force for the Synthetic Theater of War '97, with specific emphasis on the development of highly complex individual combatant synthetic forces. In exchange for research facilities and access to subject matter experts (SME) located on the Combat Center, DARPA agreed to make the emerging STOW technologies available to the Combat Center for training. DARPA further agreed to explore advanced user interface concepts at this research facility and focus the effort, to be known as CommandVu, on the live fire training conducted at the Combat Center.

The LeatherNet system is an integration of existing and developing computer tools being combined to create a dynamic, user-centered environment for Marine Corps training, mission rehearsal, and analysis at (MCAGCC). The computer tools currently implemented in LeatherNet include: Marine Corps Synthetic Forces (MCSF), a computer simulation tool that can simulate the USMC individual combatants, vehicles and behaviors; Terrain Evaluation Module (TEM), a planning tool developed by the US Army used for terrain evaluation and includes tools for line-of-sight and weapons coverage analyses of targets in terrain; CommandVu, an enhanced synthetic environment which provides three-dimensional representations of MCSF behaviors, the display of control measures, for the training of Marine Corps commanders at MCAGCC; and CommandTalk, a speech recognition and natural language understanding system that provides Marine Corps commanders with the ability to communicate

to the simulation software and MCSF entity through spoken language and pen based gestures.

The development and project management is being conducted by the Naval Command Control and Ocean Surveillance Center, Research Development Test & Evaluation Division (NRaD) on behalf of the Defense Advanced Research Projects Agency Synthetic Forces Program Manager. There is a diverse group of government agencies, academic institutions, and private industries all collaborating to develop the LeatherNet System.

This paper is organized in four main sections. First, a summary of the concept of operation of the LeatherNet; second, discussion on the development approach; finally, a description of the system and products of the LeatherNet.

3. Concept of Operation

The LeatherNet system provides Marine Corps commander with a set of tools that aid in the tactics development, briefing, simulation/rehearsal and debrief (After Action Review-AAR) of two live fire training ranges at MCAGCC. In order to enhance training and improve live fire range safety several DARPA advanced technologies are being developed in cooperation with MCAGCC. It must also be pointed out the missions and resources available to the Marine Corps make LeatherNet a microcosm for STOW-97 given the fact it includes infantry, special operations capabilities, mech/armor, air and an amphibious capability.

The intended user of LeatherNet is a Marine Corps commander at the company or battalion level of command and the immediate subordinates. The concepts and human computer interface toolkit provided in LeatherNet should be able to scale up or down the echelon of command from Platoon Leader all the way to Regimental Commander or above. It is apparent to the developers that there will need to be specific tools tailored for each different echelon of command, yet, it is believed that the basic concepts and approach will apply throughout.

Figure 3-1 depicts a Marine Corps commander using the system to formulate and develop tactics and plans by using a walk-in-synthetic-environment (WISE) as an interactive tool. Commanders can: create and issues orders to MCSF (ModSAF) entities with speech, create ModSAF command and control measure with speech, enter the simulation as a tank, helo, etc... or as a Stealth. Commanders can also access the TEM tools through window on the WISE for group planning and briefing. Subordinates are immersed in Helmet Mounted Display (only one is

currently functioning) to allow individual freedom of movement through the terrain.

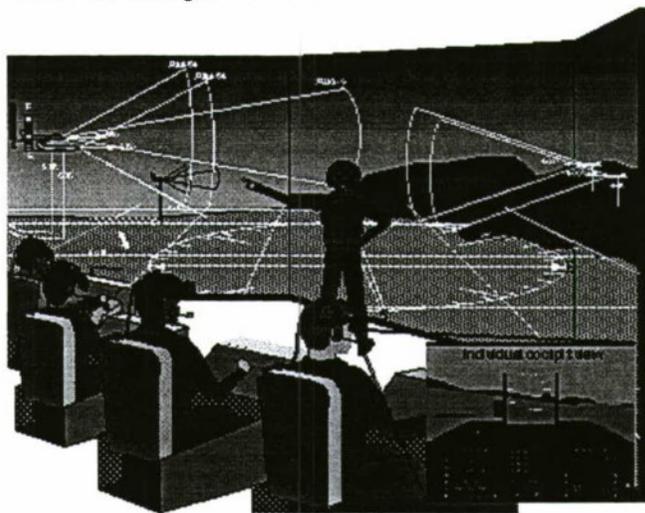


Figure 3-1 Concept of Operation

Typically, 3D “stealth” viewers in the STOW community have only been used for AAR as a “video tape” type tool. Users have only been able to passively see what has transpired. In the CommandVu concept, the environment is the tool set. 3D objects are “buttons”. Click on a 3D MIAI tank model and information about the tank will be displayed attached to the model and will move in the environment with tank. Command and Control measures have 3D representation and can be used in planning, briefing, rehearsal, or debrief to augment the user’s view of the situation. By providing an augmented environment users may be able to reinforce decision-making skills before they go the field.

In order to limit the amount of training required to learn how to use the system and to provide the user with a “natural” user interface, the system was designed with the user’s normal methods of data input and output in mind. User commands are normally issued verbally and responses are auditory. Therefore, the LeatherNet system implemented speech and gesture as a primary input method into the synthetic training environment. Additionally, a common user interface for LeatherNet tools will simplify the use of the computer system for the Marines Corps Commanders.

4. Development Concepts and Approach

4.1 Development

USMC SF Capabilities are provided as enhancements to an existing Modeling and Simulation (M&S) system called Modular Semi-Automated Forces (ModSAF). These enhancements are defined as Problem/Change Requests (PCRs) related to the USMC domain for Battalion-level entities and their tactical behaviors for Amphibious Operations. To implement these USMC capabilities, a spiral, concurrent development methodology is employed, beginning with an analysis of the M&S requirements to the validated software (see Figure 4-1.)

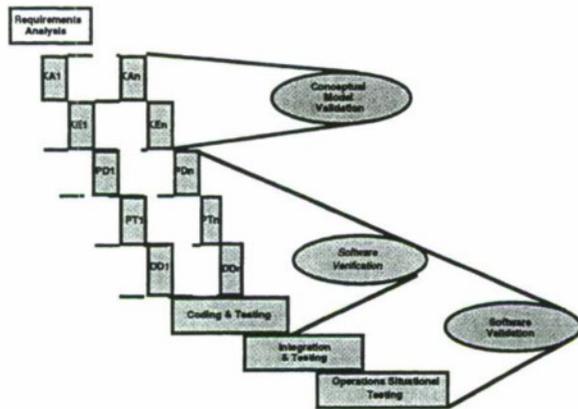


Figure 4-1 Development Process

In compliance with the principles of DoD and USN Verification, Validation, and Accreditation (VV&A), a Conceptual Model (CM) is formulated through systematic Knowledge Acquisition (KA) and Knowledge Engineering (KE) processes so that the USMC M&S Management Office (MCMSMO) can evaluate and approve the use of the capabilities for the envisioned training application and its inclusion in an M&S CM repository. To ensure adequate information is collected and analyzed for the required capabilities, the development process incrementally produces knowledge and prototypes (PT). The prototypes confirm the level of fidelity and functionality with requisite Subject Matter Experts (SMEs), thereby helping to identify missing information and the long-lead design tradeoffs (preliminary design - PT).

An integrated Development Engineering Team (DET), comprising SMEs, engineers, programmers, and testers, works together toward a validated software product on assigned domain entities throughout the development process. Acquisition

and engineering of knowledge on USMC echelons and operational doctrine continues until the prototype confirms acceptable simulated performance to cognizant SMEs (KA1 - KAn, KE1 - KEn). A preliminary design for the capability is generated (PD1) and prototyped (PT1) based on the initial KA; depending on the prototypes general acceptability, detailed design data is collected (DD1), and coding and testing activities begin. Software verification is performed by the DETs to ensure the coded and tested software product satisfies the preliminary design.

Integration testing is performed on sets of PCRs in scenarios established to exercise capabilities in a typical operational scenario and to regressively test legacy code. These activities incrementally validate the integration of capabilities developed for scheduled DARPA Combined Tests (CTs) held throughout the year. These CTs are conducted in large multi-force, multi-service scenarios, with SMEs as test operators, monitors, and evaluators. The results of these tests are then submitted to MCMSMO for accreditation of use in DoD training systems employing USMC amphibious operations.

4.2 Integration and Test

The goal of this integration effort is to ensure the MCSF/CFOR, CommandVu, CommandTalk and Synthetic Environment are integrated into a seamless training environment that will be a successful tool to train JTF Commanders and USMC Commanders. Concurrent with this goal is the intent of LeatherNet to provide MCAGCC with a virtual synthetic environment system capable of augmenting and enhancing the existing MCAGCC Combined Arms Exercise (CAX) training syllabus.

Integration and test team (ITT), comprising SMEs, programmers, systems administrators, systems integrators, and testers combine verified software from individual DETs for milestone-driven integration tests, and configuration management. The milestone-phased testing approach of the integration strategy emphasizes the operational, technical, and systems operations of each subsystem integration and test/demonstration. This strategy also allows scheduling control to accommodate STOW-97- and MCAGCC-driven mission-based MCSF/CFOR requirements throughout the LeatherNet development period. Each integration milestones maps directly to specific MCSF/CFOR mission capabilities. Once the integration test is successfully completed, integrated software is then baselined and labeled by the local configuration manager as a formal LeatherNet baseline. Then MCSF software is checked into the Synthetic Forces configuration management system called Version Integration

Control System Semi-Automated Forces / Open Semi-Automated Forces (VICS SAF/Open SAF).

LeatherNet is integrated at MCAGCC on a monthly basis. A developmental integration suit is setup at NRaD so preliminary integration and testing may be performed. The LeatherNet development suit is designed and developed for various rapid prototypes as well as experiments. This integration suit is used to debug and improve the final MCAGCC target system. NRaD utilizes a commercially available revision control tool (CVS) to streamline integration and upgrade to new versions of the software and system. The ITT is responsible for integration and installation of integrated LeatherNet system at the MCAGCC LeatherNet Lab.

5. LeatherNet System Description and Products

The major components of the LeatherNet system are briefly described in the next sections.

5.1 Marine Corps Synthetic Forces (MCSF)

The Marine Corps Synthetic Forces (MCSF) component of the LeatherNet project, will provide a representation of Marine Corps platforms and behaviors operating in a realistic synthetic environment for STOW-97 and beyond. MCSF is based on ModSAF. The desired resultant capabilities of this system are the integrated system functions and behaviors to accurately represent a Marine Expeditionary Force (MEF) Forward composed of a Ground Combat Element (GCE), an Air Combat Elements (ACE), and a limited Combat Service Support Element (CSSE), to operate in a joint synthetic theater of war. It will also provide accurate, concise behavioral representation of specialized teams, functions, and specific mission areas, including an amphibious assault, movement-to-contact, attack, consolidation, defense, and patrolling. LeatherNet will also be capable of representing in software a platoon leader, company commander, and command staff that can provide command and control of the integrated system functions and mission areas through the DARPA sponsored Command Forces (CFOR) project.

MCSF individual combatants entities are modeled down to the individual combatant, with unit tasking down to the Fire Team composed of four synthetic Marines. Other individual combatant entities include the Rifle Squad, Machine Gun Teams, Assault Teams, and Mortar Teams. Vehicles include variants of the Amphibious Assault Vehicle (AAV), Light Armored Vehicle (LAV), and the High-Mobility, Multi-purpose Wheeled Vehicle (HMMWV), and the

M1A1 main battle tank. Aircraft include the CH46E, CH53E, AH-1W, AV-8B and F/A-18.

MCSF will continue the development and refinement of individual combatants, ground and air vehicles, systems, and related behaviors for MEF-(Forward). The main focus of this effort will be the development of advanced behaviors to accomplish the Marine Corps amphibious assault and attack missions in a realistic environment. This will include Marine Corps CFOR representation in software of a Rifle Platoon Leader, Rifle Company Commander, and various command assets to enable command and control of synthetic Marine Corps forces in performing the desired missions.

Also a more realistic, dynamic synthetic environment will be developed to meet the requirements of MCSF. Current high resolution triangulated irregular networked terrain efforts will need to be expanded to include other terrain databases in which individual combatant involvement is required. MCSF will investigate the need to be able to react and represent the effects of this environment within the behavior of its vehicles, individual combatants, their systems, and the command and control elements. Representation may include effects caused by the environment resulting in changes to performance of individual combatants, sensors, communications, weapons systems, and vehicle operations. MCSF is especially in need of a representation of the littoral area for Amphibious operations in the surf zone.

5.2 Terrain Evaluation Module (TEM)

The current version of TEM installed at the LeatherNet Lab is TEM 7.2. Current TEM is use in a standalone configuration, but ability to take TEM data, such as weapons fan coverage's and line-of-site calculations into the 3D CommandVu environment in underway. This would be an example of a prototype of a 3D-C4I device for planning in a simulated training environment.

5.3 CommandVu

As the Human-Computer Interface (HCI) development effort of LeatherNet, CommandVu must support four closely related tasks. The first task is to support the development of tactical expertise among USMC field officers in the context of live fire exercises at MCAGCC. The second task is to support the development and validation of MCSF algorithms. The third task is to provide an interface that will allow Marine Corps commanders to interact with the MCSF units by using natural inputs including speech and gesture commands. And the last task is

to serve as the Marine Corps interface for participation in STOW-97.

These applications require a robust, easy-to-use interface which can support both individual decision-making behavior and team coordination in a simulated outdoor environment, where both real and computer-generated forces are led through tactical battle problems. This HCI design combines a variety of diverse technologies and performance improvement concepts in novel ways, called a "concurrent" virtual environment. Such an environment is typified by:

- Simultaneous use of head-mounted displays (HMD), multiple large-screen (CAVE) displays, and conventional CRT displays, which are selected as a function of the user role or a performance improvement objective, and which may be changed repeatedly during a single simulator session.
- Support of multi-mode input technologies, including gesture, speech, physical control panels and virtual control panels, and pointers. Input methods will be highly redundant, to support user preference and to enhance system recognition of user intent (e.g., through simultaneous voice and gesture combinations).
- Support of individual and coordinated activity between several live participants interacting with intelligent computer-generated agents (both vehicles and humans controlled by ModSAF and MCSF), involving both vehicle operation and dismounted tasks (including navigation through the environment on foot). Participants may work at different levels of representation, from different perspectives in the environment, and for different specialized goals.

Development of CommandVu involves essentially all of the analysis, design, and evaluation tasks of most complex system programs, however some issues have proven unique for both the application and the user. As a result, some of the design approaches taken for this program have required specialized supporting research in human factors and performance measurement which deserve discussion. In particular, efforts include:

- an iterative design and review program is being pursued with prospective users for task domain guidance but, more importantly, to foster an understanding of the potential utility of state-of-the-art technologies for solving current problems in new ways.
- a series of studies concerning perceptual problems in navigating rapidly through large expanses of simulated terrain, especially when

one individual controls the perspective for other participants, in real time.

- a research program in perceptual and cognitive integration of multiple visual perspectives, supported by the CommandVu system, and of orientation problems associated with rapid perceptual or display changes (e.g. HMD to CAVE and back).
- a research effort to develop "augmented visualization," or artificial cues which can enhance decision-making and which can encourage effective team collaborations in real time.
- an examination of adaptive performance improvement methods for gradually removing artificial cues from displays until the environment matches that experienced in the real world, with performance maintained at the desired level.
- an empirical study of display resolution requirements, to minimize computational loads while retaining task-relevant realism and user confidence.
- design and evaluation of optimal system configuration and control tools -- including efficient allocation of functions to physical and virtual control devices -- to support rapid realization of desired scenes and actions. This work is focused on requirements to set up and demonstrate scenarios quickly, to expedite training, and to examine the nature of team collaborations by observing how these tools are used.
- efforts to exploit the potential of comprehensive performance measurement in complex scenarios, to provide real time performance feedback and to conduct long term "trend" analysis and decision modeling through examination of cumulative databases.

CommandVu display devices include, helmet mounted displays, large projection displays arranged as a walk through environment with three dimensional sound.

Multi-modal input devices include, speech recognition, traditional keyboards and mouse, virtual on-screen buttons, wireless three 3D gyro mouse, on screen controls (tape displays), joysticks and tracking devices.

CommandVu includes the use of selectable 3D command and control measure, including: vehicle trails (blue for friendly, red for enemy), lines, polygons and text. Figure 5-1 depicts a USMC mechanized movement to contact.

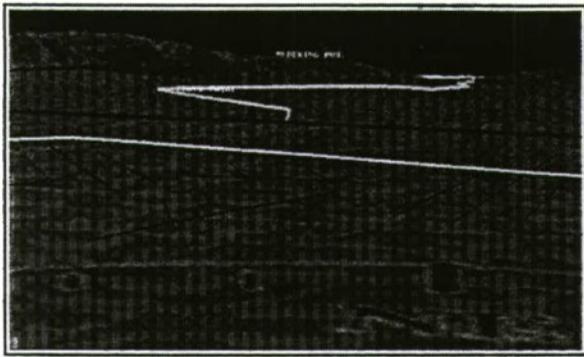


Figure 5-1 Command and Control Measures

Phase lines, an assemble area, a blocking position, checkpoints and boundary lines are all represented in the CommandVu environment. These 3D control measures are linked to their counterparts in MCSF. If a checkpoint is selected and moved in the 3D environment, the checkpoint on the MCSF Plan View Display (PVD) will move simultaneously.

Weapons fan can be represented in CommandVu in a similar manner to the 3D command and control measures. Figure 5-2 shows an MIA1 weapons fan on R400 terrain at MCAGCC.

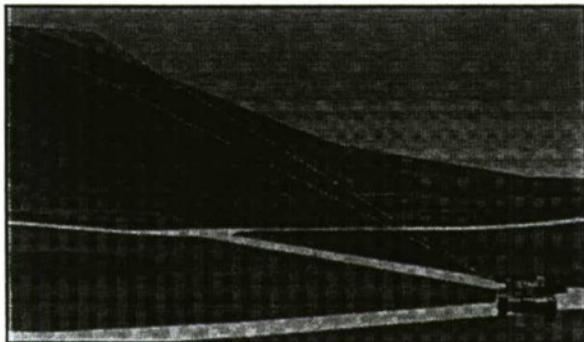


Figure 5-2 Weapons Fan

5.4 CommandTalk

Initial efforts were focused on the development of speech to MCSF, with current work focused on the development of speech, the Command Control Simulation Interfaced Language (CCSIL) developed by MITRE, improved user feedback, increased vocabulary for the Platoon Leader, Company Commanders, and Battalion Commanders, speech to CommandVu, and the addition of the ability to speech and gesture to the system at the same time.

Technical Capabilities include a 500+ word vocabulary, featuring “radio-speak” phraseology (use of entity call-signs), ability to control vehicles keyed by unit call signs (regardless of unit composition),

ability to control individual combatants, ability to set control measures, issue “op-orders”, and “frag-orders”.

There are four benefits of adding human language capabilities to the LeatherNet system. First is the ability to create forces and control measures. Second, is the ability to assign missions and task frames to synthetic forces. Third is the he ability to control and modify missions during program execution. And forth, is the ability to control system functions (such as fundamental PVD controls).

6. Conclusions

LeatherNet has made significant progress in its two years of existence and withstood intense scrutiny by the Office of the Secretary of Defense, Marine Corp Flag officers including the Commandant of the Marine Corps as well as the Defense Director of Research and Engineering.

The ability of the LeatherNet Team to produce rapid result is directly attributable to several factors briefly explained below.

Do not hold a “not invented here” attitude. LeatherNet built MCSF based on DARPA’s ModSAF and applied the systems and behaviors of the other DARPA service Synthetic Forces development effort to build MCSF ground vehicles and Aircraft. CommandVu built its Human Computer Interface concept on top of the existing NPSNET software developed by the Naval Postgraduate School, Monterey, CA. The addition of other DARPA-sponsored effort in speech and gesture technologies were evaluated and integrated into the core system.

Adopt a flexible development approach that could be adapted to the changing STOW requirements as well as the MCAGCC user evolving requirement. When LeatherNet was first introduced to the Marine Corps, MCAGCC did not have operation requirement nor an accepting attitude. But within the first 6 months the USMC and MCAGCC leadership recognized the value of the DARPA relationship and the potential benefits for MCAGCC training. As the Marine Corp interest and training needs grew, so did the use of the LeatherNet Lab, and the need for specific user define requirements.

Set a clearly defined vision. The specifics of what features, entities and behaviors were added to the system to date were constantly being refined, but the core vision remains virtually untouched from the LeatherNet kickoff. LeatherNet also challenge some of the brightest developers in the community (NRAD,

HRL, MITRE, ATI, NPS, SRI, OGI, KES, GSC and BMH) to pull together and achieve a common vision.

User-centered development. Another key factor in the success of the LeatherNet project was fact that the Lab was placed on a Marine Corps Training Base that supports the Fleet Marine Force, vice in a research environment. LeatherNet followed the lead of another DARPA-sponsored project call WISSARD (What If Simulation System for Advanced Research and Development) and funded strong site manager support to interface with the Marine Corps and the DARPA developers. Additionally, the ability to work closely the users and understand their needs as it applies to the mission accomplishment has proven to be invaluable.

7. Acknowledgment

The authors would like to thank our DARPA sponsor, CDR P. A. Feldmann, USN, Synthetic Forces Program Manager, for guidance and support.

8. References

- Osga, Glenn and Murray, Steven (1994). *Preliminary Design Document. Concept of Operations: CyberView Human Computer-Interface*. San Diego, CA: Naval Command, Control and Ocean Surveillance Center, RDT&E Division.
- Berkowitz, J. (1995a) *Technical and Scientific Report: Human Computer Interaction (HCI) Design Guidelines and Concepts for LeatherNet*. San Diego, CA: Galaxy Scientific Corporation for the Naval Command Control and Ocean Surveillance Center.
- Berkowitz, J. (1995b) *Technical and Scientific Report: LeatherNet Human Computer Interaction (HCI) Functional Design*. San Diego, CA: Galaxy Scientific Corporation for the Naval Command Control and Ocean Surveillance Center.
- Berkowitz, J. (1996) "Considerations for the Use of Entity-based Simulation for Tactical Decision-Making Training", the 6th Computer Generated Forces and Behavioral Representation Conference Proceedings.

9. Author's Biography

Jeff Clarkson is a project manager and engineer with NCCOSC RDTE DIV, San Diego. His current responsibilities include the project management and development of the LeatherNet System, MCSF,

CommandVu and CommandTalk in support of the DARPA STOW-97. Previously, he was a Naval Aviator flying CH-46 Helicopters out of NAS North Island, San Diego, and NAS Norfolk, VA. Mr. Clarkson has an Engineer's Degree in Aerospace Engineering and a M.S. degree in Aerospace Engineering from the Naval Postgraduate School. He did his undergraduate work at Georgia Tech. His interests are in the areas of Individual Combatant Computer Generated Forces, Human System Interaction and the application of 3D Environments towards Human Computer Interfaces.

John Yi is a Senior Systems Engineer/Integrator with Koam Engineering Systems (KES). His current responsibilities include the design, development, integration and testing of LeatherNet System. Previously, he was involved in the design and development of Intelligence Analysis System (IAS) at MCTSSA, and Real-time simulation of Advanced Air Traffic Control at Naval Research and Development. Mr. Yi has a B.S. degree in computer science from UC San Diego, and is currently working on his graduate degree in Information Management from UC Irvine. His interests are in the areas of real-time, and faster than real-time simulations, and common operating/data environments.



Computer Generation of Joint Theater Missile Defense (TMD) Assets

Donald E. Carver and George M. Parsons
Missile Defense Battle Integration Center
U.S. Army Space and Strategic Defense Command
P. O. Box 1500
Huntsville, Alabama 35807-3801

Dr. William T. Naff
BDM Federal, Inc.
950 Explorer Boulevard
Huntsville, Alabama 35806-2808

1. Abstract

The Missile Defense Battle Integration Center (MDBIC) will provide computer generation of the complete set of joint TMD assets via the Extended Air Defense Testbed (EADTB) augmented by Modular Semi-Automated Forces (ModSAF). The MDBIC has demonstrated the capability of the EADTB and ModSAF, operating in tandem through distributed interactive simulation (DIS), to model all elements of the TMD "pillars" in real time.

The EADTB is a state-of-the art constructive simulation with DIS compliance demonstrated at the prototype level and scheduled for delivery in September 1996. A High Level Architecture (HLA) translator is also under development. The rapidly expanding EADTB capability allows simulated entities such as weapon systems to be modeled independently as objects on the gameboard. These "models within the model" can be selected, modified, or developed by the user.

The MDBIC manages a master library that currently contains a complete set of extended air defense system models. All three services have initiated development of TMD active-defense models to be validated and certified by the system proponent offices for specific ranges of applications. The MDBIC will thus augment any DIS or HLA exercise with a full suite of computer-generated joint TMD assets including proponent-certified active-defense system entities.

2. Introduction

The Missile Defense Battle Integration Center (MDBIC) will provide computer generation of the complete set of joint Theater Missile Defense (TMD) assets via the Extended Air Defense Testbed (EADTB) augmented by Modular Semi-Automated Forces (ModSAF). TMD active defense assets will be based on validated models, certified by the Army, Navy, and Air Force system-proponent offices. The MDBIC has demonstrated the capability of the EADTB and ModSAF, operating in tandem through distributed interactive simulation (DIS), to model all elements of the TMD "pillars" in real time.

3. The Pillars of Theater Missile Defense

The theater missile threat includes both tactical ballistic missiles and cruise missiles. TMD includes contributions from four "pillars": active defense; passive defense; attack operations; and battle management/command, control, communication, computers, and intelligence (BM/C⁴I). While active defense includes all means for killing theater ballistic missiles (TBMs) in flight, passive defense comprises all measures to make TBM targets harder to find and harder to kill. Attack operations are the offensive actions taken to kill TBMs and their supporting infrastructure on the ground. BM/C⁴I, which supports all of the activities associated with the other three pillars, is often represented as a foundation rather than a pillar, as shown in Figure 1.

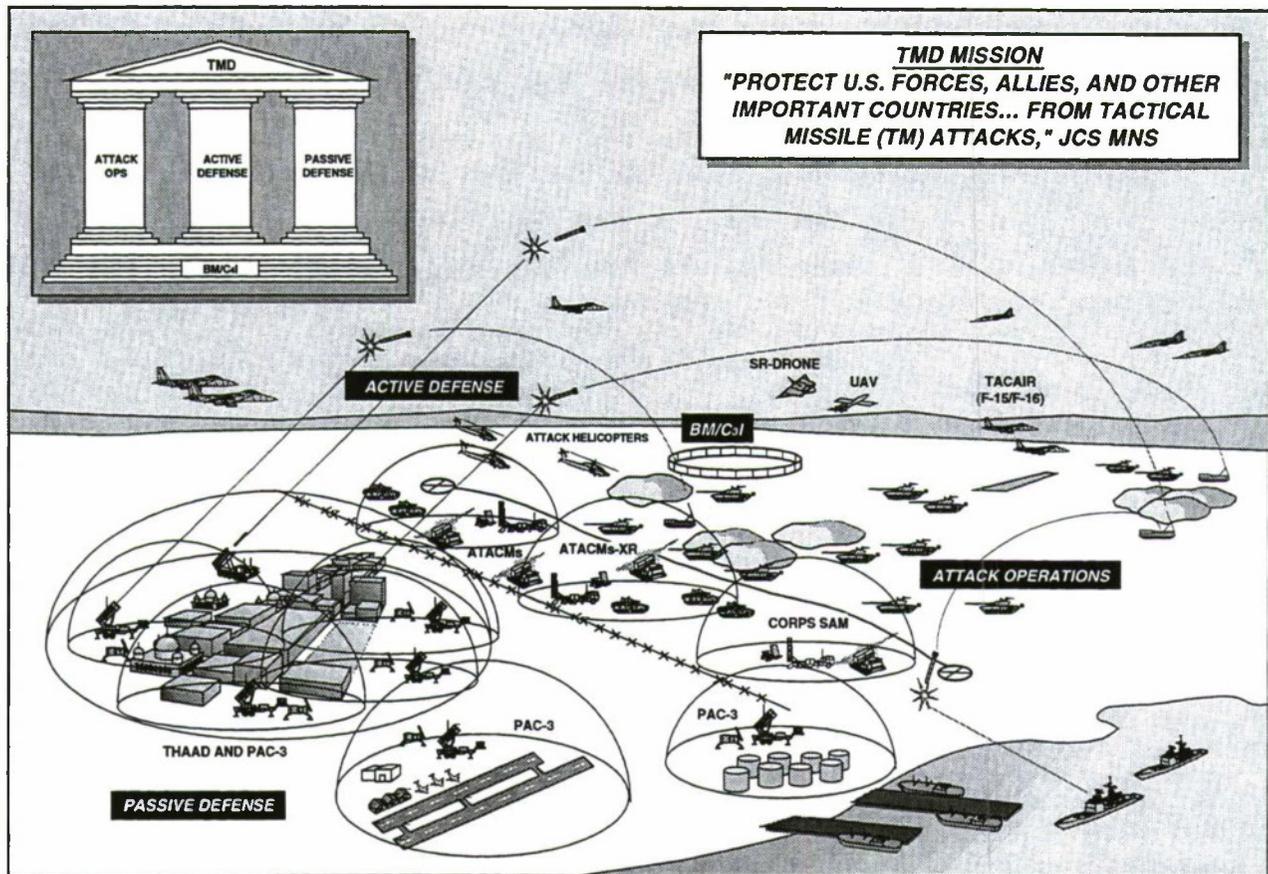


Figure 1. The Pillars of TMD.

While most new system acquisitions, such as THAAD, and major system upgrades, such as PATRIOT PAC-3, are associated with active defense, the requirements on these systems and the total success of TMD depend on the integrated performance of all four pillars.

4. The Extended Air Defense Testbed (EADTB)

4.1 EADTB Functions

The EADTB, a state-of-the-art constructive simulation, will use DIS [and later, High Level Architecture (HLA)] to provide active defense elements including THAAD, PATRIOT, Corps SAM, and AEGIS. As required, the EADTB can generate theater missile threats as well.

The EADTB allows simulated entities, such as weapon systems, to be modeled independently as objects on a gameboard (Figure 2). These “models-within-the-model,” which are referred to as Specific System Representations (SSRs), can be selected, modified, or developed by EADTB users. Thus, the EADTB serves as a model-development environment as well as a model in itself.

SSRs include the representation of rule-set-based “thinkers,” both human and machine, that react to perceived data. The “thinker” portion of each model is formally restricted from access to truth data. The analyst has full access to the rule sets controlling “thinker” behavior through the SSR code, allowing “what ifs” on rule set logic. However, any change of a PEO-approved model requires renaming, thus ensuring the integrity of certified models.

The EADTB gameboard environment includes elevation based on Digital Terrain Elevation Data (DTED), features based on Digital Feature Analysis Data (DFAD), time-varying weather (with clouds), and both infrared (IR) and radio frequency (RF) backgrounds. The gameboard reference coordinate system for DIS entity-state specification is Earth-centered rotating (WGS-84).

The EADTB has been recognized as an ideal framework for joint-service and international creation, certification, and sharing of models. The MDBIC maintains a master library of SSRs for joint system studies. Efforts are

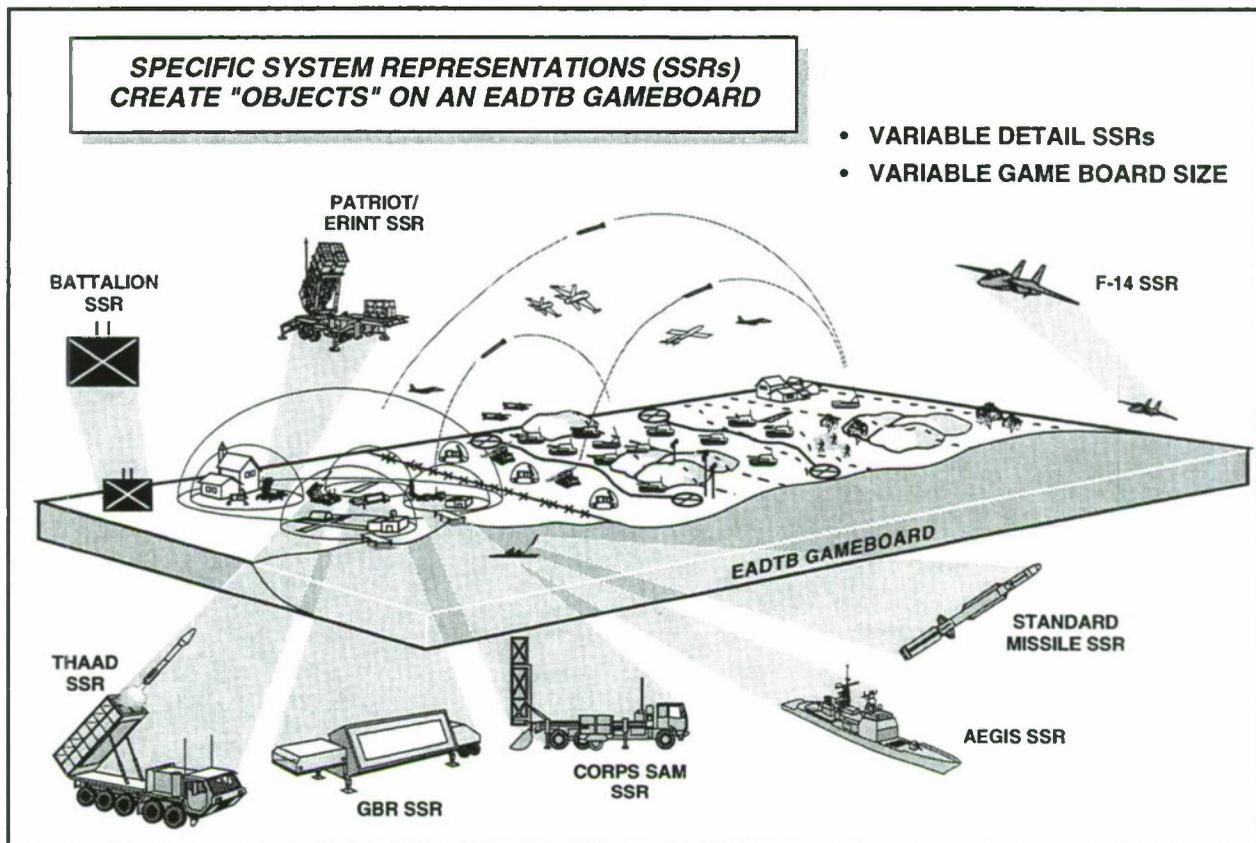


Figure 2. The Specific System Representation (SSR)

JR-R1-6190-B02

currently underway by Army, Navy, and Air Force participants to certify models of key systems. Figure 3 illustrates this concept in which proponents develop experimental models for their own use while preparing and certifying models for use by the community.

A key part of this "sharing-of-models" concept is the documentation of certification. The documentation will specify how verification and validation were accomplished and will state the limits of model validity. Thus, the certifying agency can bound the applicability of the models and protect itself from model misuse.

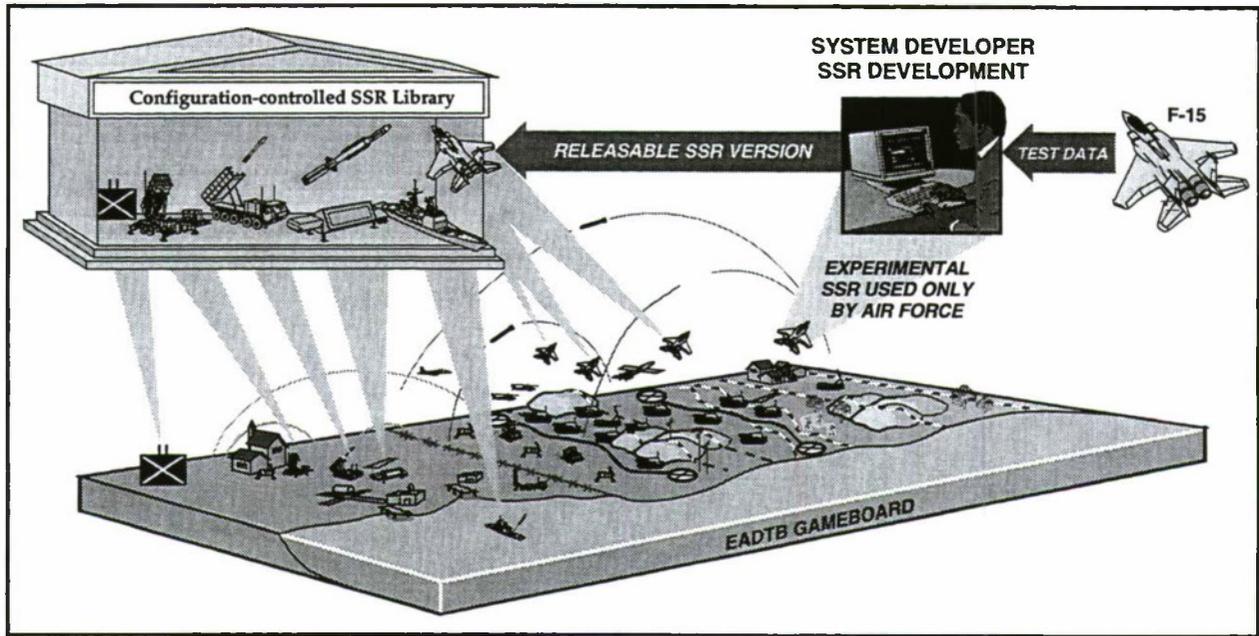
4.2 EADTB History and Siting

From its beginning in 1989, EADTB development has been managed by the Testbed Product Office (TPO) within the U.S. Army Space and Strategic Defense Command (USASSDC). Funded by the Ballistic Missile Defense Organization (BMDO), the EADTB is intended for joint-service, international use, with the primary goal of serving the extended air defense community. Initial EADTB sites (see

Figure 4), which are sometimes referred to as nodes, were located at the USASSDC Advanced Research Center (ARC) in Huntsville, Alabama; the NATO Strategic Headquarters Allied Powers Europe (SHAPE) Technical Centre (STC) in the Netherlands; and the U.S. Army Air Defense Artillery School (USAADASCH) at Fort Bliss, Texas. Since the first incremental capability delivery in 1994, sites have been added at the Joint National Test Facility (JNTF) in Colorado; the Naval Surface Warfare Center (NSWC) at Dahlgren, Virginia; the Tactical Air Command and Control Simulation Facility (TACCSF) at Albuquerque, New Mexico; and the Ballistic Missile Defense Organization (BMDO) and the Warfighter Analysis and Integration Center (WAIC), both in the Washington, D.C., area. Memoranda of Agreement (MOAs) are currently under negotiation for the addition of sites in France and Germany.

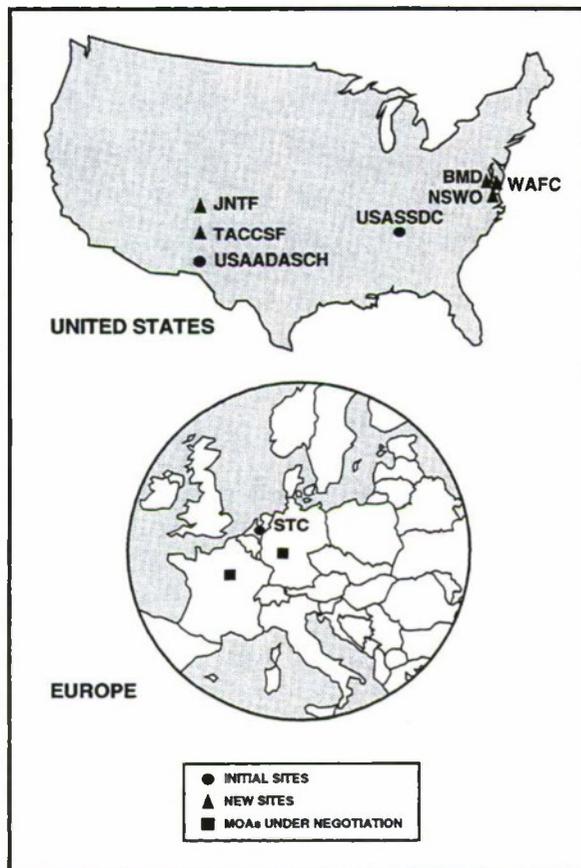
5. DIS/HLA Compliance

As illustrated in Figure 5, the EADTB adapts naturally to the DIS environment. Ghost SSRs



JR-R1-6190-B03

Figure 3. The EADTB Library



JR-R1-6190-B04

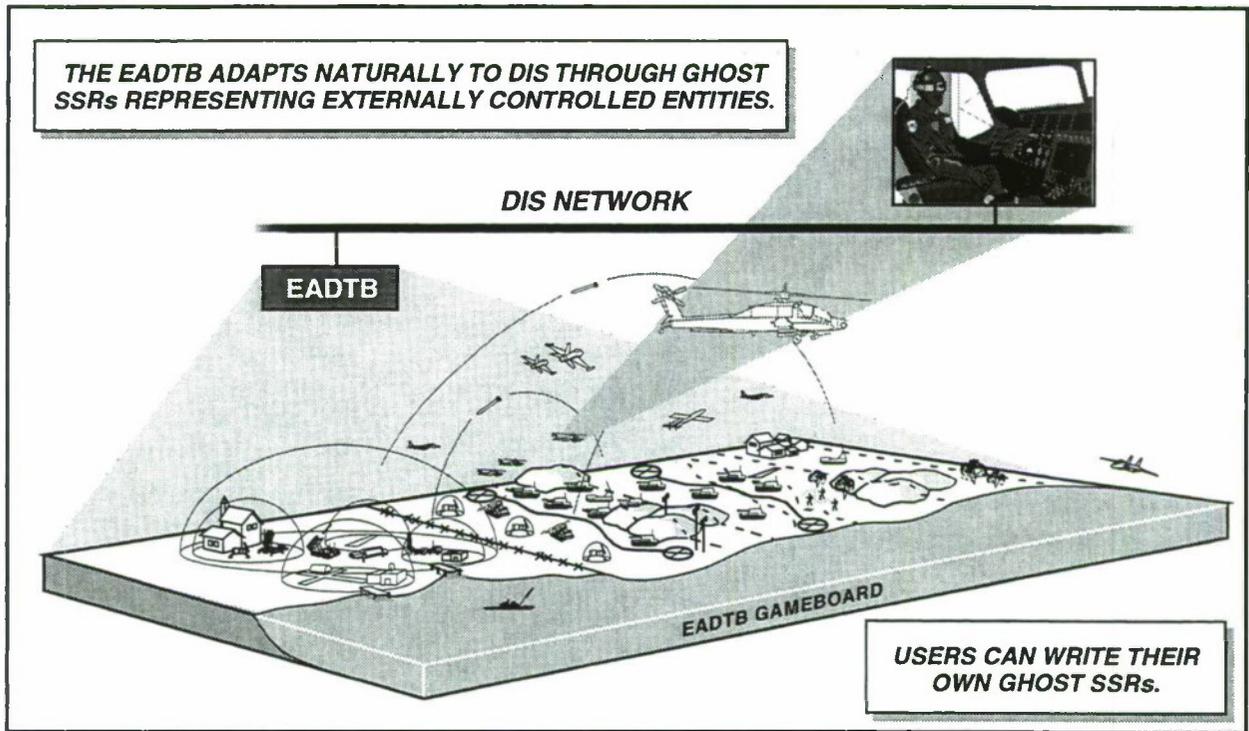
Figure 4. EADTB Sites

simply replace standard SSRs to allow external control of simulated entities. Users can create their own ghost SSRs in order to introduce new externally controlled entities into the paradigm.

A prototype Distributed Interactive Simulation (DIS) compliance capability was delivered and successfully demonstrated in September 1995.

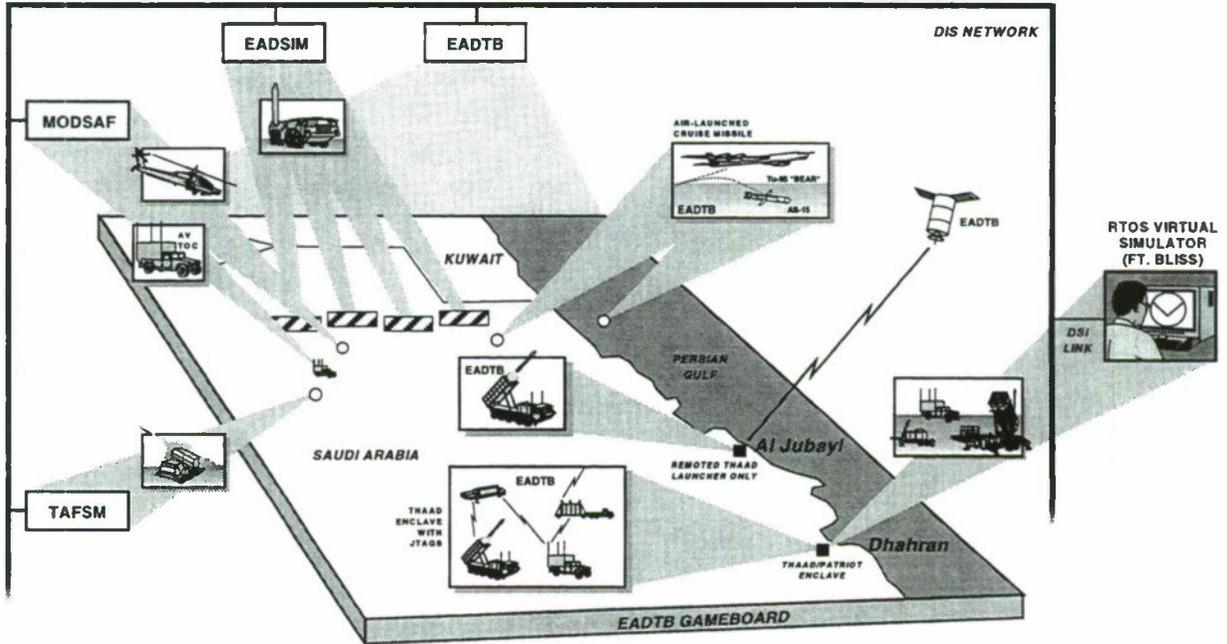
Figure 6 shows the prototype DIS demo scenario, which has been frequently run at the MDBIC. ModSAF, the Extended Air Defense Simulation (EADSIM), and the Target Acquisition Fire Support Model (TAFSM), along with the EADTB, were housed at the MDBIC's Advanced Research Center (ARC). The Reconfigurable Tactical Operations Simulator (RTOS), an operator-in-the-loop virtual simulator, was configured to represent a PATRIOT battery, sited at Fort Bliss and linked to the ARC via the Defense Simulation Internet (DSI). (A video teleconferencing link between the ARC and Fort Bliss was maintained simultaneously over the DSI to view live operator and on-screen activity.)

TAFSM supplied field artillery attack operations assets including ATACMS and an artillery tactical operations center (TOC). ModSAF supplied Army aviation attack operations assets consisting of attack helicopters and an aviation TOC. EADSIM provided the TBM threats and their transporter erector launchers (TELs). The EADTB generated the cruise missile



RB-R1-6190-H01

Figure 5. The EADTB Natural Interface with Other DIS Participants.



JS-R1-6190-E01

Figure 6. EADTB DIS Demonstration in September 1995

threats and the bombers that launched them; the Defense Satellite Program (DSP) space-based sensors and Joint Tactical Ground Station (JTAGS); the THAAD radar, launcher, and missiles; and an Air Defense TOC (ADTOC). The exercise, which included a total of 165 simulated entities, was intended to demonstrate the EADTB potential for populating DIS exercises with constructive TMD-asset entities.

The first delivered, DIS-compliant version of the EADTB will be capable of simulating 80 types of entities and will send and receive entity-state, signal, transmit, start/resume, and stop/freeze PDUs.

6. Capability for TMD Computer Generated Forces (CGF)

The EADTB will have its first delivery of a DIS-compliant version (Version 4) in late September 1996. A DIS-to-HLA translator is also under development to provide a near-term, limited HLA capability to support STOW 97.

The Version 4 delivery will allow entity-state PDU generation for all of the approximately 80 different distinct types of entities represented by the existing set of EADTB SSRs. (The total number of entities instantiated and played simultaneously in the EADTB is constrained only by computer power.) Users can, as stated above, create new SSRs for internally controlled and/or externally controlled (i.e., ghost) entities.

Simulated entities include sensor, missile, and launcher components of PAC-2, PAC-3, and THAAD systems; fighters and bombers; airborne, space-based, and surface-based sensors; jammers; cruise missiles, ballistic missiles, and ARMs; and TMD-capable cruisers. As stated previously, the EADTB maintains a formal partition between perceived data and truth data. CGF entities contributed by the EADTB will react to three classes of perceived data:

- Information received via signal PDUs
- Information received via messages from EADTB-controlled entities
- Information from on-board sensing capability

Special capabilities include the launch of an internally controlled TBM by an externally controlled launcher. This capability was added to support interoperability

with high-fidelity land-combat models. Thus, the land-combat simulator can control the movement of the launcher and the generation of the launch event. The EADTB constrains the missile to move with the launcher until the launch event occurs, after which the EADTB controls missile movement.

The concept for EADTB support to exercises is illustrated in Figure 7. The EADTB can support a federation of live, virtual, and/or constructive simulations by populating the scenario with computer-generated TMD assets. The EADTB SSR library will be the source of individual entity models comprising simulations validated by system proponent offices.

7. Summary

The mission of the MDBIC includes support of training, mission rehearsal, operations analysis, combat development, and materiel acquisition. Computer generation of TMD assets via the EADTB and other models is a key means of supporting these activities. The EADTB library of proponent-validated entity simulations will provide a quality capability for computer generation of joint, four-pillar TMD assets. For information on the level of detail of specific models available, please contact the authors.

8. Authors' Biographies

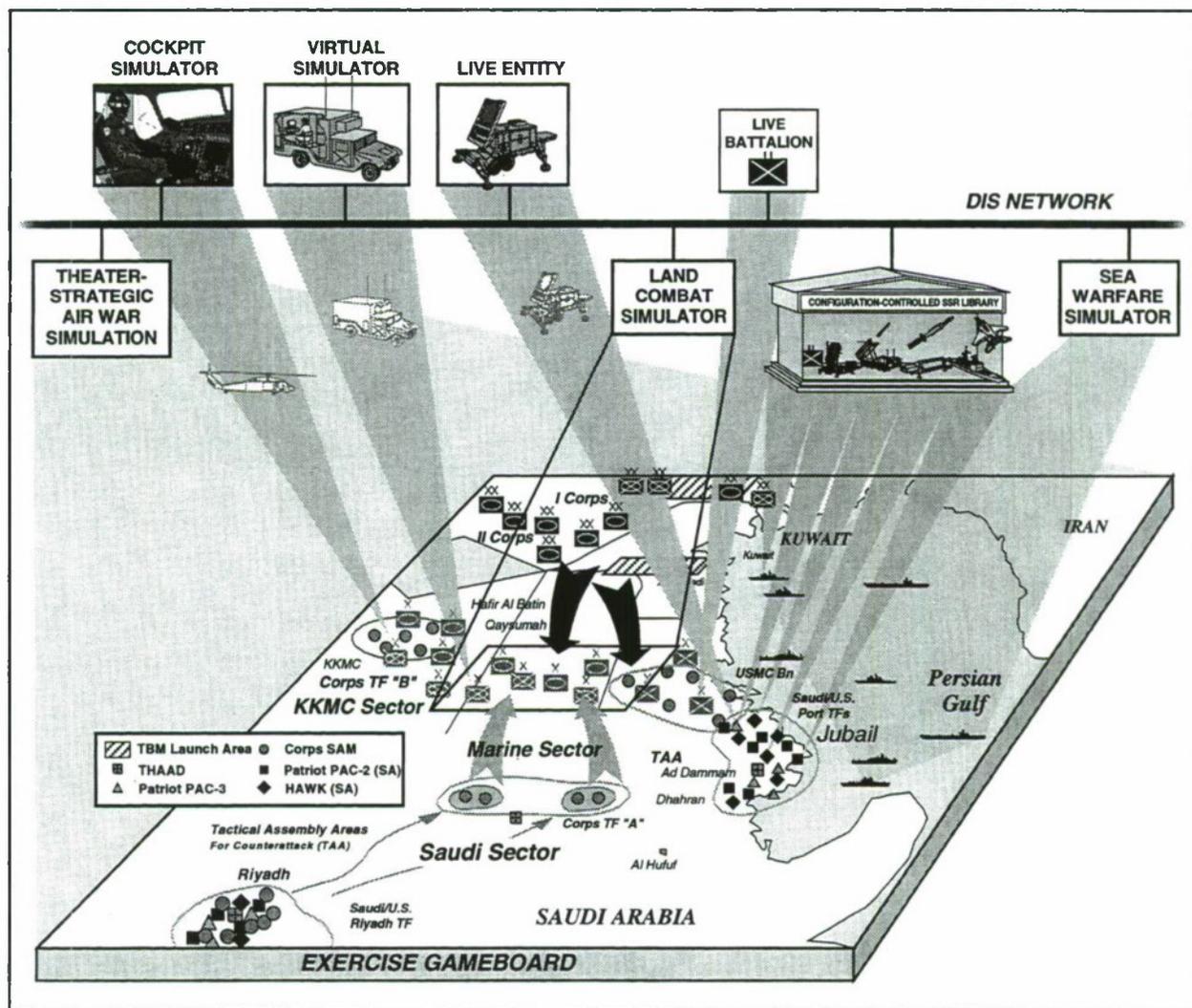
Don Carver has 12 years experience in defense-related systems simulation including serving as chairman of the CROSSBOW-S Digital Simulation Steering Group, chairman of the Hardware Committee of the BMDO Computer Resources Working Group, and lead engineer for development of the USASSDC's Advanced Research Center computer resource requirements for support of missile defense analysis. Mr. Carver's recent interests and activities include application of simulations for training and analysis to serve the warfighter. As general engineer for the MDBIC, he currently leads an effort to link the EADTB with the Corps Battle Simulator (CBS). Mr. Carver has an M.S. degree in Chemical Engineering.

Moody Parsons is Acting Chief Engineer for the Testbed Product Office of the USASSDC Missile Defense Battle Integration Center (MDBIC) in Huntsville, Alabama. He is responsible for program management activities for EADTB software development. Previous assignments include program management for Directed Energy Weapons Space

Experimentation at USASDC; Supervisor of Modifications/Special Projects, Tennessee Valley Authority Sequoyah Nuclear Plant; and other assignments dealing with nuclear and solar power generation. Mr. Parson's current interests include development of advanced simulation tools and networking strategies for integration into federations to serve modern warfighter needs. He has a B.S.E. degree in Engineering with Electrical Option.

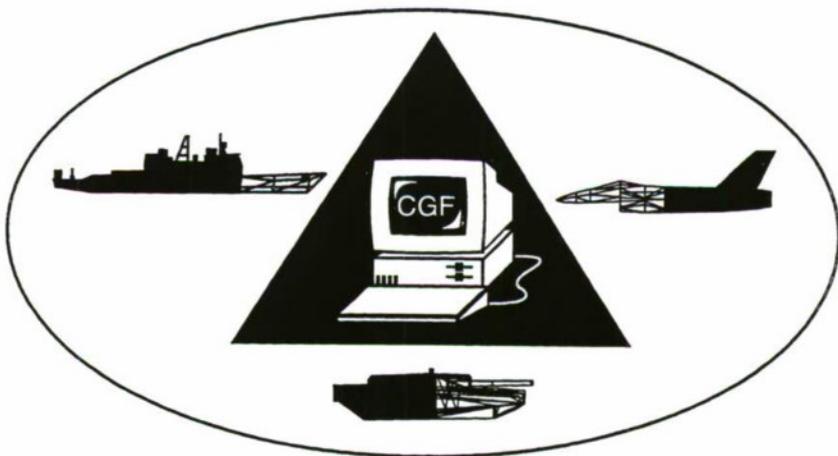
Tim Naff has 25 years experience in defense modeling and simulation ranging from technology and phenomenology modeling to military operations

research at the raid, battle, and campaign levels. His model development and application experience includes live, virtual, and constructive simulations comprising test-range activities, laboratory hardware-in-the-loop, and man-in-the-loop as well as all-digital simulations. Dr. Naff, who is Director for Systems Analysis at BDM, recently developed and applied a campaign-level model of TMD attack operations supporting an Army COEA effort. His current interests and activities also include DIS and HLA model confederation planning and development. Dr. Naff has a Ph.D. in Physics.



JS-R1-6190-A01

Figure 7. CGF Representing TMD Assets Drawn from EADTB Library of Validated Models.



The JPSD Corps Level Computer Generated Forces (CLCGF) System Project Update 1996

Jeffrey C. Peacock, Jr. Kevin C. Bombardier, James Panagos
SAIC

20 Burlington Mall Rd
Burlington MA. 01803

jpeacock@bos.saic.com, kbombard@bos.saic.com, jpanagos@bos.saic.com

Thomas E. Johnson
Raytheon Company
528 Boston Post Rd
Sudbury, MA 01776
tej@swl.msd.ray.com

1. Abstract

The Corps Level Computer Generated Forces (CLCGF) system is being jointly developed by SAIC and Raytheon for the Joint Precision Strike Demonstration Program (JPSD). CLCGF is a system that is centered on the linkage of the constructive, aggregate-level simulation Eagle, with the virtual, entity-level simulation ModSAF. The purpose of this paper is to update the community on enhancements that have been made to the CLCGF system over the last year. The following topics will be covered; resolution management, proxy, Distributed Eagle, tactical message processing, new SIU (Simulation Interface Unit) client applications, and the development of a North Korean rocket launcher threat.

As with any constructive - virtual linkage, Aggregation/Disaggregation is the main reason for creating the system. Until now CLCGF has supported two methods for controlling Aggregation/Disaggregation: Operator selection from the GUI, and Call For Fire events. While these two techniques supported the initial needs of JPSD they are clearly limited. As a result we have designed a Resolution Management library (libresman) which extends the Aggregation / Disaggregation Triggers available. The idea is to provide a framework for supporting any number of user defined triggers. These triggers are then monitored by the resolution manager and resolution changes are effected automatically. The application of aggregation/disaggregation triggers are performed in two ways: via a data file specific to a given scenario, or by dynamic events which occur during a given simulation (ex. Call For Fire).

In preparation for the 1995 JPSD exercise CLCGF was enhanced with the capability to update Eagle with status information on disaggregated units controlled by non ModSAF based simulations. This

new capability, called "proxy" was implemented as a new state of disaggregation whereby a linkage between an aggregate unit and remotely reported non ModSAF based entities was made. A second more interesting capability was also added. Through the proxy technique the capability to attach sensor components to remote vehicles was also developed. This allowed the JPSD program to represent a live UAV in the simulation environment which was capable of producing Reconnaissance Exploitation Report (ReccExRep) reports based on the state of Aggregate units

Distributed Eagle was developed by TRAC Ft. Leavenworth as an extension to the existing Eagle model. Distributed Eagle's main purpose is to increase the size of the aggregate battle being simulated by the CLCGF system. During the JPSD 95 demonstration CLCGF ran over 300 aggregate units which began to push the limits of Eagle, and the Eagle <-> SIU linkage. As a result TRAC developed the concept of Distributed Eagle which sought to increase the size of the Aggregate battle by distributing the battle across multiple platforms.

One of JPSD's many thrusts has been the integration of constructive, virtual and live systems. JPSD's version of ModSAF 2.0 has been modified to allow the simulation to process a subset of the "tactical messages" which are produced by live/simulated systems used by JPSD. To accomplish this we have added new libraries to ModSAF which support tactical message processing. The following tactical messages are currently processed; Reconnaissance Exploitation Report (ReccExRep), JPSD Sensor Control messages, and Fire Mission Call For Fire (FM;CFF) messages.

In addition to Eagle, the current SIU has been used to support 2 other programs; the STOW exercise generation effort and the Rapid Battlefield Visualization (RBV) program.

2. Introduction

2.1 The JPSD Program

One of the Joint Precision Strike Demonstration (JPSD) program's goal is to introduce and implement new technologies into the defense arena that can address and correct precision strike deficiencies. To facilitate this goal, the JPSD program has created a simulation environment which is used to evaluate technologies, train users, and perform experiments necessary to reduce sensor-to-shooter timelines. As part of this environment, the JPSD program has sponsored the construction of the Corps Level Computer Generated Forces (CLCGF) system.

The primary purpose of CLCGF is to provide the corps level simulation environment for DIS exercises in which the above mentioned program goals can be met. CLCGF is used during the JPSD exercises to simulate maneuver and artillery units contained in an Army corps. The simulated units provide stimulus for and interact with tactical hardware systems and their operators.

2.1.1 JPSD 1996

The simulation being developed for JPSD 1996 is based on a North Korean Multiple Rocket Launcher threat. CLCGF will portray the North Korean OPFOR, as well as friendly airborne sensor platforms at the entity level. In addition, CLCGF will also play over 600 aggregate units to fill out the battlefield. The aggregate units will provide additional targets for sensor systems and will be available for disaggregation based on fire missions executed during the scenario.

2.2 The CLCGF System

Entity-level simulations represent each entity which exists on the virtual battlefield at the individual platform level. They typically represent entities from the individual platform level up to the company level. They use the DIS protocol to interact with other entity-level simulations, and simulate the physical characteristics of each entity to determine battlefield outcomes. On the other hand, constructive simulations represent groups of entities as single, aggregate unit objects. They typically represent units at the company or battalion level up to the division or corps level. They are typically not designed to interact with other simulations, but instead simulate the entire battlefield internally, and use Monte-Carlo techniques to determine battlefield results.

The DIS environment has traditionally included only entity-level simulations. It has provided a sound environment for small-scale, tactical troop training, as well as a potential testbed for evaluating new vehicles and weapon systems. However, simulating the effects of entity-level simulations in corps level operations has remained beyond the reach of the DIS environment, due to network bandwidth and computer resource constraints. Using current network and computer technology, a traditional DIS exercise is simply not capable of supporting a corps level operation. This was the primary motivation for creating a CLCGF which utilizes both constructive and entity-level simulations. Transmission of unit state data at the aggregate level is a key factor which decreases network load by significantly decreasing the number of PDUs transmitted in a large-scale exercise. If DIS is to support a 100,000 entity exercise, representation of some units on the battlefield as aggregates is likely.

The simulation engine of the CLCGF has been built by integrating the constructive, aggregate-level simulation Eagle, with the virtual, entity-level simulation ModSAF. This simulation engine interacts with various live, tactical hardware systems, including: the Reconfigurable Workstation (RCW), the Maneuver Control System / Phoenix (MCS/P), and the Automated Deep Operations Coordination System (ADOCS).

In order to allow military training and analysis of scenarios of interest to JPSD, the CLCGF must generate a full corps-level exercise. To accomplish this goal, many technical challenges need to be addressed. These involve issues such as efficient incorporation of aggregate units into DIS, effective incorporation of DIS entity-level information into constructive simulations, development of a dynamic aggregation/disaggregation protocol, interaction between constructive and entity-level simulations, and interaction between a constructive/virtual simulation, live systems, and engineering-level simulations. The work performed on the CLCGF to date has focused on these fundamental goals.

2.3 CLCGF Interaction with other JPSD Systems

In order to create a test and evaluation environment in which to conduct JPSD experiments and studies, the requisite constructive, virtual, and engineering-level simulations must inter-operate with one another, as well as with current and future fielded, tactical systems used in Army Corps operations. A block diagram of the CLCGF system and the non-DIS systems with which it interfaces is shown in Figure 1.

The CLCGF system consists of the linkage between Eagle, the SIU (Simulation Interface Unit - whose primary function is to link Eagle to the DIS network), and ModSAF. It is responsible for simulating the entities and aggregate units on the corps battlefield, presenting a plan view display,

interacting with other DIS simulations, and interfacing with the other non-DIS systems shown in the block diagram. The RCW and MCS/P systems are used to present a picture of the tactical battlefield situation to an operator (via intelligence feeds), and to initiate precision strike target nominations. The ADOCS system is used to create, monitor, and assign fire missions to corps artillery assets. The STRIKE simulation is used to simulate the deployment, fly-out, and impact of smart submunitions. The Tactical Gateway (TGW) is responsible for routing tactical messages to other platforms which have registered for specific types of tactical messages. For details on the interactions of the systems described in Figure 1 see (Calder et. al 1995).

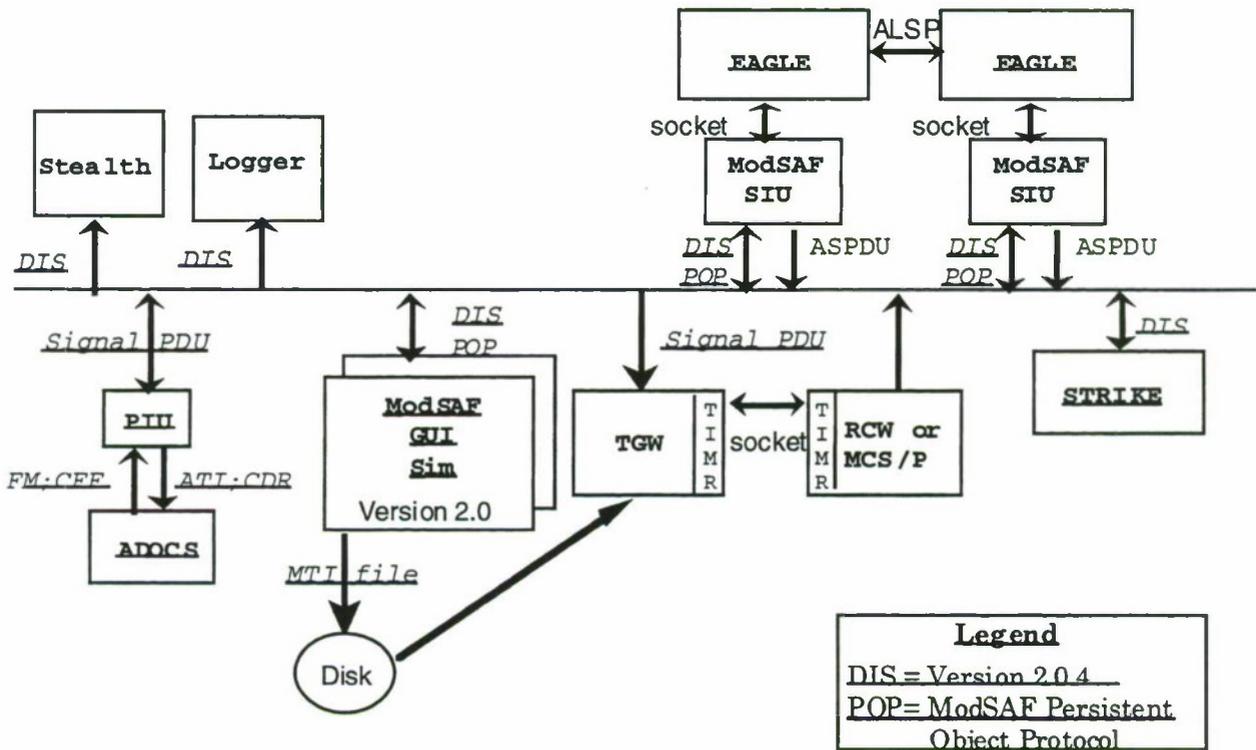


Figure 1: CLCGF Interface Block Diagram

3.1 Resolution Management

3.1.1 Motivation

CLCGF uses a variety of mechanisms to implement resolution change. However, a more general approach is desirable, where resolution decisions are made by the software. It should be possible for the operator to define rules that would allow the system to aggregate/disaggregate automatically. In order to be effective this module must allow the resolution change rules to be modified at any time during the exercise so that aggregation/disaggregation can be based on the evolution of the scenario. Such a module would allow designers to create complex logic for controlling resolution change. In addition, such a module would centralize resolution management and reduce code duplication.

In response, a resolution management library has been created (*libresman*). *Libresman* provides a general "clearing house" for all aggregation and disaggregation decisions in CLCGF. The motivation for this library is to:

- Centralize the aggregation/disaggregation decision logic.
- Provide a basic discipline of resolution change (which can be automated) so that more sophisticated behaviors can be created.
- Define a precision in the semantics of aggregation and disaggregation that uses a wider range of properties to calculate resolution change. (i.e. disaggregate units that are not only within range-as is typical of current technology, but that may have a particular weapon system etc.)

The last two factors impart a control over resolution change which is offered as a tool to counter the dreaded "spreading disaggregation" problem that can plague systems of varying simulation fidelity.

3.1.2 Implementation

Libresman determines whether an entity, vehicle or unit, should change its resolution state. It is designed to abstract the aggregation or disaggregation decision as a boolean outcome; as such, complex boolean algebra can be built into the decision-making process. The domain of the resolution change decision is defined at run-time and its range is determined by the designer but can be potentially the entire ModSAF address space (i.e. any library - subject to the usual C compile and link restrictions).

An entity must be registered with *libresman* via the function *resman_register_aggregate*. With this simple call to *resman_register_aggregate*, an entity acquires a basic set of rules which govern its resolution. *Libresman* will automatically define a "negation" rule for any rule defined. This negation rule will undo the effect of a rule when its predicate is no longer applicable. *Libresman* maintains its own VTAB of entities which have rules associated with them. Each of these are ticked with respect to all entities known to ModSAF, local and remote. Upon each tick, the appropriate list of resolution rules (based on the entity's current resolution state) is evaluated. If any rule is found to be true, a resolution change will be implemented for the entity.

Defining an entity's resolution change behavior is controlled by defining its resolution change rules. A resolution change rule contains the following:

- 1) A pre-defined predicate.
- 2) Information on whether to use the rule to re-aggregate or disaggregate; this information is maintained by the system and is transparent to the user.
- 3) Information on the types of rules to "respond to". Using this information, a rule can be defined to be fired if and only if another rule has previously been fired. This allows for fine-tuning pairs or groups of rules so that they can work together.

In general, there are two ways to associate a rule to an entity: by reader file or by function call. Reader file association allows resolution management to be defined on a per scenario basis. Reader rules are read once, (at startup) and thus are very useful in defining blanket (i.e. global) rules. On the contrary, function call association can be made at any time during the execution of the code thus enabling registration of rules in response to changing scenario events.

Currently, resolution change predicates include:

- *unit_in_area*. If the aggregate enters a specific area, disaggregate.
- *duration*. If the aggregate has been disaggregated for some period of time, re-aggregate.
- *sphere_of_influence*. If another type of vehicle comes in proximity to us, disaggregate.(currently not implemented).

These are but a few of the predicates that could be defined, the above mentioned serve as test cases. The definition of additional predicates is left to the user and will be driven by new requirements as they develop.

3.1.3 Adding new predicates

Predicates can be chosen from a pre-defined set or can be defined at compile time. A newly defined predicate must be a new boolean function with 2 arguments: the vehicle id of the owning entity, and a parameter structure defining private (to the predicate) information. The predicate then has to be declared in libresman's list of predicates. Once defined, the new predicate can be used via reader file or function call.

3.2 Proxy

3.2.1 Motivation

In a typical CLCGF scenario, Eagle maintains control of aggregated units until a request for disaggregation is received by the SIU. Upon receipt of the disaggregation request, Eagle relinquishes control of the aggregate to the SIU, and then periodically polls the SIU for updates, such that Eagle can maintain current positional and compositional data for the disaggregated unit. The SIU initiates the disaggregation and subsequent creation of the entities comprising the aggregated unit. To accomplish a successful disaggregation, ModSAFs operating on the same exercise id and PO database as the SIU assume the responsibility of simulating the entities. Entity state information is then summarized and used to update Eagle as to the state of the aggregate. This method posed a limitation that all entities comprising an aggregate must be simulated within ModSAF, thus limiting the potential sources of entity level simulation for Eagle aggregates.

Libproxy was developed to allow non-ModSAF based entities to influence the outcome of the aggregate battle being played in Eagle. Eagle was originally designed to run as a stand-alone application without a DIS interface. The linkage with the SIU has indirectly given Eagle a limited DIS interface. The interface required to allow Eagle to utilize entities reported from remote sources on the DIS network must utilize information common to both the ESPDU and the Eagle-SIU interface. To accomplish this the proxy interface was designed to utilize the unit designation within Eagle and the marking contained within the 'entity-marking' field of the ESPDU.

An additional capability which extends the proxy interface, provides CLCGF ModSAF with the capability to attach sensor components to remote vehicles. This capability allowed for the representation of a live UAV in the simulation environment, which was capable of generating

Reconnaissance Exploitation Reports(ReccExrep) for aggregate units. The need for this capability was two fold; one the UAV platforms were not being played by ModSAF and two, the CLCGF ModSAF is the only system that understands how to generate the entity level representation for an aggregate unit.

3.2.2 Implementation

Initialization of libproxy occurs for all CLCGF ModSAF's during startup. Libproxy is initialized with a ModSAF reader file, in which a mapping between the Eagle unit designation is made with the expected 'base' marking of the entities being reported via the ESPDU. During initialization, a proxy table is created, storing the aggregate marking with the expected base marking for each entry in the reader file. If the proxied unit is to simulate a sensor, the sensor name must also be included within the reader file. This table provides a means by which either the SIU or the proxy machine may query libproxy to ascertain the status of a proxied unit.

As with a disaggregation command, when a unit is commanded to be proxied, the SIU assumes responsibility for providing the information necessary to keep Eagle updated as to the status of the proxied unit. However, unlike the process of disaggregation, the SIU does not initiate the creation of the entities comprising the aggregate, but instead the SIU utilizes the data received from remote entities being reported via the ESPDU to provide the inputs required to update Eagle. The SIU commences the proxy process by querying libproxy to determine if this unit was specified as a proxy unit in the reader file. If this unit was identified as a proxy, the SIU extracts the 'base' marking for the remote entities, and begins a search of the main vehicle table (VTAB), looking for REMOTE_VEHICLES which contain the specified 'base' marking. Upon finding a match, the SIU updates the proxy table with the vehicle id and entity-marking of the received remote. Since aggregates typically are defined to contain more than one vehicle, the SIU does not consider a unit proxied until remote entities representing each constituent of the aggregate have been found in the vehicle table, i.e. if the Eagle aggregate is composed of 10 entities, the SIU must find 10 remote vehicles, each having a unique marking comprised of the base marking, to complete the proxy of this unit. If Eagle requests a status update on a unit, for which not all constituents have been remotely simulated, the SIU reports back to Eagle the last state of the aggregate. Upon completion of the proxy, i.e. all constituents of the aggregate are represented as remote entities, the SIU creates an average representation of the positions and velocities of the remote entities, and reports this average back to Eagle upon request. Each time Eagle

requests a status update on a proxied unit a check is made to insure that all remote entities comprising the proxied unit are still active. If any constituent is not currently being simulated the SIU reverts to reporting the last state of the aggregate, and begins the search for the constituents of the aggregate again.

A goal of the JPSD program is to integrate live tactical forces into the simulation environment. For the JPSD 95 demonstration, a live UAV was introduced into the exercise by means of telemetry data transmitted to a ground station. The ground stations translated the coordinates of the UAV from CONUS coordinates to the simulated battlefield in Korea and generated Entity State PDU's, providing a method whereby the live UAV was incorporated into the exercise. Use of the proxy technique allowed the attachment of a sensor component to the remote entity.

To accomplish this task a dedicated ModSAF was initialized with a new command line argument, '-proxy', which permitted tagging remote entities as proxy vehicles within libremote. On the machine performing the proxy, libremote queries libproxy to determine if the received entity is a proxy vehicle. This check occurs only on the initial reporting of an entity to libremote i.e., the entity has not yet been assigned a vehicle id. If the vehicle is identified as a proxy, libremote tags the entity as VTAB_REMOTE_PROXY and adds the vehicle to a proxy tick list, allowing us to tick the vehicle similar to a remote but with the additional sensor components added.

Upon determination that a remote is to perform a proxy mission, a check is made to retrieve the sensor component to tick from the proxy table. As with local vehicle processing, proxy utilizes the parameters files defined for the simulation of each vehicle. Within these files, more than one sensor component may be specified for the vehicle. To insure that only the desired sensor was activated, all defined sensor components are initially deactivated, then based on the specified component in the proxy table, the specific sensor is activated

3.3 Distributed Eagle

Eagle was initially designed to simulate units at the battalion-level and higher. CLCGF requires that units which are candidates for disaggregation be simulated at the company/battery level. Eagle scenario developers have made the necessary accommodations to support the CLCGF effort. However in doing so they have increased the unit count in Eagle to the point where run speed has become an issue. During JPSD 1995 the Eagle

scenario consisted of approximately 300 units. In order for Eagle to maintain real-time while connected to the SIU, the Eagle time step was changed to 3 minutes. Clearly a performance improvement was necessary for the FY 1996 demonstration, which was planning a 600+ unit Eagle scenario. The solution was to develop 'Distributed Eagle'.

Distributed Eagle seeks to increase the size of scenarios it can support by dividing up its scenario among multiple platforms. Distributed Eagle communicates over a local area network, through the use of ALSP. Distributed Eagle was developed by TRAC Fort Leavenworth as an enhancement to the existing model. Its main purpose has been to increase the size of the Aggregate battle.

The Aggregate Level Simulation Protocol (ALSP) was designed to permit multiple, pre-existing warfare simulations to interact with each other over local or wide area networks. In concept it was patterned after SIMNET (now Distributed Interactive Simulation - DIS) where each simulation controls its own objects and shares information about them with other simulations. The advantage to this type of protocol is that aggregate level constructive simulations which represent distinct segments of a battlefield can be connected and thus effectively provide a common environment to support major training exercises.

For the JPSD 1996 Demonstration the planned Eagle scenario has 600+ units which will be simulated by two Eagle <-> SIU combinations running in parallel. Up to four Eagles can be connected via ALSP. The CLCGF system has been designed such that various combinations of Eagles and SIUs can be supported.

3.4 Tactical Message Processing

CLCGF's version of ModSAF 2.0 has been modified to process a subset of the "tactical messages" which are produced by live/simulated systems used by JPSD. To accomplish the tactical message processing two new libraries have been added to ModSAF. These new libraries support processing for the following tactical messages; Reconnaissance Exploitation Report (ReccExRep), JPSD Sensor Control messages, and Fire Mission Call For Fire (FM;CFF) messages.

All tactical messages used by JPSD are sent wrapped within a Signal PDU. The new libraries added to ModSAF to support control of and processing for tactical messages are libtactmsgctrl and libtactmsg. Libtactmsgctrl provides the user with a GUI that allows processing of the supported messages to be enabled/disabled. In addition, the operator is able to select which radio nets should be listened to.

Libactmsg is responsible for registering a callback with libduproc to catch all Signal PDUs. The callback function consults libactmsgcntrl to determine which types of messages and which radio nets are to be processed. Given that a message passes these tests it is then dispatched to a parser to determine message type and contents.

The ReccExrep message contains a sensor report being transmitted by one of the UAVs. Each UAV is assigned it's own PO overlay in which to store PO PointClass objects. A new PointClass object is created for each ReccExrep received. The operator is free to show/hide or even delete any overlay using the ModSAF overlay editor. This capability will allow JPSD to capture where each UAV is reporting targets and determine how accurate the intelligence reports are with respect to ground truth. Having the ability to view ground truth and perceived truth (intelligence picture) on a single system is a very valuable tool which should provide additional insight into how the scenario is unfolding.

The JPSD Sensor Control message was developed to provide an operator with a mechanism for controlling the flight path of simulated UAVs with out having to use the ModSAF PVD. This capability provides the operators at a tactical workstation with the capability to, for example, task a UAV to fly over a specific area for Bomb Damage Assessment (BDA) or to confirm an enemy location prior to generating a call for fire message. As Sensor Control messages are received the designated UAV is given a new route by creating a new PO LineClass object using the given waypoints. Once the new route has been created the PO object id for the current route is replaced with the new object id. The existing ModSAF fly route behavior continuously monitors the route and automatically re-directs the UAV based on the new route information. In addition to route modification the Sensor Control PDU is also used to enable/disable the UAVs sensor.

The JPSD FM;CFF is issued by the ADOCS to initiate a fire mission. The SIU is responsible for taking the following actions when an FM;CFF is received. Disaggregation of the target area, and disaggregation of the shooter if the mission is being issued to an aggregate unit under CLCGFs control. Note that disaggregation of the shooter is optional and in fact TAFSM will be responsible for all MLRS firings during JPSD 1996.

3.5 New SIU Client Applications

The original interface between the SIU and Eagle consisted of approximately 6 commands which utilized RPC and shared memory as the communication mechanism. During JPSD 1995 we found that the RPC/shared memory interface was unreliable during periods of heavy network traffic. As a result a decision was made to use TCP/IP sockets to interface the two systems. The socket interface has worked very well to date and given us the reliability that was lacking during 1995.

During 1996 two additional SIU clients have been developed in addition to Eagle; the Simulation Planning Agent(SPA), and the Scenario and Infrastructure Analysis Tool (SAT/IAT). The SPA is being developed under the Rapid Battlefield Visualization program, while the SAT/IAT is being developed under the STOW Exercise Implementation (XI) effort. (Juliano et. al. 1996)

3.5.1 Simulation Planning Agent (SPA)

The purpose of the SPA is to interface the SIU with MCS/P. (See Figure 2) The current system is utilized in 2 modes; as a wargaming tool, and as a visualization tool.

As a war gaming tool the SPA extracts a snap shot of the MCS/P database, parses it and forwards the relevant information to the SIU via the socket interface. The SIU then instantiates each of the aggregate units and begins transmitting Aggregate State PDUs. (ASPDU) At this point a user sitting in front of a ModSAF PVD can disaggregate units of his/her choosing and run "what if" type scenarios using ModSAF. As the scenario is executing the SPA is probing the SIU for the status of each disaggregated unit. The SPA then updates the MCS/P overlay with the status of the war game.

As a visualization tool the SPA continuously extracts unit information from the MCS/P and forwards it to the SIU. The SIU in turn creates the given units and begins transmitting ASPDUs. This capability will allow the commander to visualize the current state of both friendly and enemy troops based on current intelligence information. The current system utilizes the ModStealth to render the 3D view.

The ability to run what if scenarios and to visualize the current state of the troops is sure to provide the commander with a powerful decision aid.

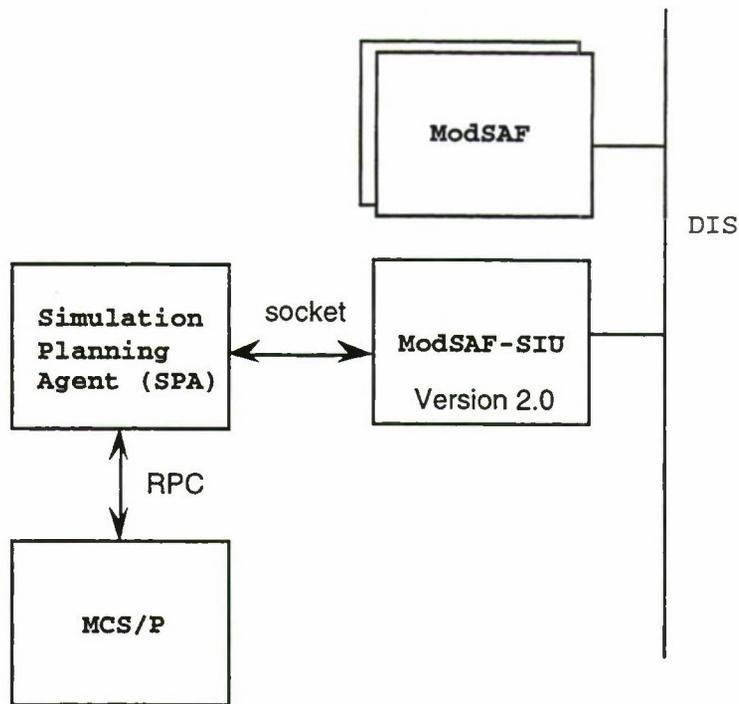


Figure 2: SPA Interface Block Diagram

3.5.2 Scenario Analysis Tool / Infrastructure Analysis Tool (SAT/IAT)

The Scenario and Infrastructure Analysis tools (SAT/IAT) are currently being developed by SAIC under the SEID contract to support STOW (Synthetic Theater of War) Exercise Implementation (XI) system. The simulation is responsible for aiding in the exercise generation, management and analysis process for Computer Generated Forces (CGF). Since future CGF exercises, including STOW, are required to support a distributed exercise of a large number of entities, a pre-exercise faster than real-time determination of scenario, network and computational validity is necessary. The SAT/IAT simulation executes at a rate on the order of several hundred times faster than real-time.

The current system consists of the SAT/IAT application with an optional interface to the CLCGF SIU. (See Figure 3) Again the socket interface originally developed to allow ModSAF and Eagle to communicate has been reused. The SIU has two uses under the exercise generation effort; as a PVD to visualize the status of a SAT/IAT scenario, and as GUI for creating SAT/IAT scenarios.

As a PVD the SAT/IAT simply requests that the SIU create each unit in its scenario so a visual representation of the scenario will appear on the ModSAF PVD. Once the units have been created by the SIU the SAT/IAT scenario is executed. During the execution phase the SAT/IAT continuously updates the SIU with the latest unit information. The ModSAF PVD gives the user the ability to visualize how the forces are positioned as well as an indication of each units strength.

As a GUI for creating SAT/IAT scenarios the ModSAF PVD is used to lay down forces and assign behaviors for Aggregate units (currently move is the only supported behavior). The operator uses the current ModSAF unit editor and execution matrix to create and task aggregate units. Once the operator has completed development of a SAT scenario he/she selects the "SAT/IAT Load Scenario" option under the "File" pulldown on the ModSAF PVD. The "SAT/IAT Load Scenario" feature then cycles through the PO database extracting all relevant scenario information. The relevant scenario information is then output as ASCII text in the format required to be used as input to the SAT/IAT application.

By interfacing the SAT/IAT with the SIU the Exercise Implementation effort has obtained several significant capabilities; a PVD for visualization, and a planning GUI for scenario generation.

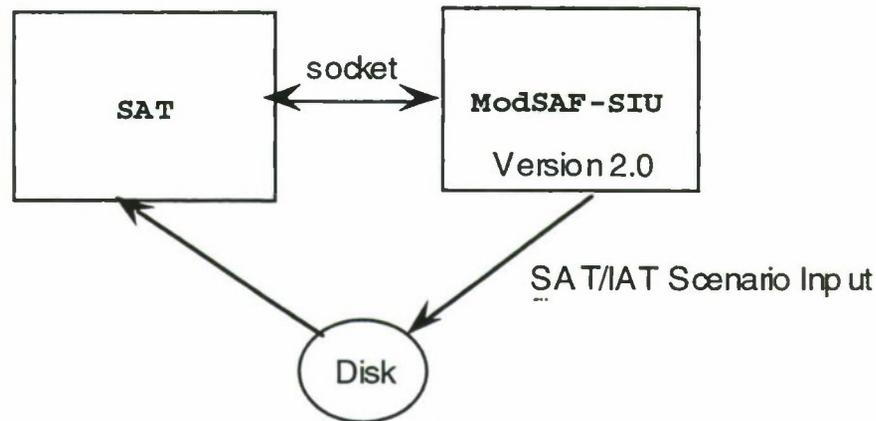


Figure 3: SAT/IAT Interface Block Diagram

3.6 North Korean Rocket Launcher threat

For the JPSD 1996 demonstration, the CLCGF ModSAF will be used to portray a North Korean Multiple Rocket Launcher (MRL) threat. The new threat behavior is being developed as a company level task which reused a significant portions of several existing ModSAF tasks. A key area of investigation for JPSD 96 is the effect of the destruction of logistics support on the overall tempo of the battle. The new MRL task will permit an in-depth analysis of the affects of eliminating the capability to re-supply shooters. The new behaviors, modeled at the company level, provide enhanced capabilities over the existing MLRS and service station behaviors currently incorporated within ModSAF. These new capabilities include:

- Ability to conceal MRL and re-supply vehicles in caves to avoid detection by airborne sensors or other intelligence gathering assets.
- Automatic route generation utilizing the defined road network if available.
- Modeling of company level re-supply behaviors.

As experience in the Gulf War proved, tracking and ultimately destroying the SCUD launchers proved to be a very difficult task. The tactics of hide, shoot, and hide proved to be a very effective technique in eluding the enemy and subsequently surviving to fight again. The MRL task currently being developed provides the same basic ability. The MRL task is designed to search an area around the initial unit placement for caves, which are reported via Entity State PDUs. At initialization of the MRL task, a determination of unit placement is made. This determination compares the locations of the vehicles comprising the unit, with the reported caves. For

units located within the confines of a cave, the task sets the concealment bit (bit 19) for each vehicle in the unit, thereby hiding the unit from airborne sensors and protecting the vehicles from the effects of the majority of weapon systems which would be employed. While in a concealed state, the only way a vehicle can be damaged is for the cave to sustain damage, thereby trapping the concealed vehicles in the cave.

Once the determination of concealment and possible damage has been made, vehicle level movement is utilized to move the undamaged vehicles from the cave which causes them to become visible. Once all vehicles have exited the cave, a route is computed to move the company from the current unit location to the designated fire location, using the road networks defined in the terrain database. A company march task is assigned to the unit, tasking it to follow the generated route. When the unit arrives at the fire location, the unit aligns along a user defined firing azimuth. Set-up timers, defined for the type of shooters are used to simulate the time necessary to prepare the unit to fire. Upon termination of the set-up timers, the unit performs a coordinated barrage attack against the target area, firing a user specified number of rounds. Upon conclusion of the fire mission, the unit begins a tear-down process, where tear-down timers simulate the time required to prepare the unit to travel. At the expiration of these timers, a new route, generated as before, takes the unit from the current location to the defined re-supply point, utilizing the road network if possible. The unit is tasked again with a company march and proceeds to the re-supply point.

Upon arrival at the re-supply point, the MRL task determines if re-supply is possible. Re-supply is defined as possible if the following conditions are true:

- The re-supply unit is not damaged.

- The re-supply unit is not moving, i.e. the designated re-supply vehicles are at the re-supply point.
- The re-supply unit is within range of the re-supply point, i.e. the re-supply vehicles have not been stopped due to damage.

If any of the conditions for re-supply fail, the MRL task will not attempt to perform the re-supply. In the case where all the re-supply vehicles for a battery are damaged, the MRL task will compute a route back to the initial starting location of the unit and the behavior will terminate. In the other cases, i.e. the re-supply vehicles are moving, or the re-supply vehicles are not within range, the MRL task will wait at the re-supply point for a user defined period. During this wait cycle, the MRL task will check the conditions for re-supply, and if the conditions are met for re-supply, the company service station task will be spawned. If after the expiration of the wait timer, the re-supply vehicles still have not reached the re-supply point, the MRL task will compute a route back to the initial start location of the unit and terminate the behavior for the company.

The MRL task makes several coarse level checks of the current situation prior to spawning the company level service station task. These checks are in place to determine if the conditions are such that a re-supply is even possible. If re-supply is deemed impossible due to a missed rendezvous or damage to re-supply vehicles the company service station task is not spawned and the scenario continues without a re-supply cycle. As stated earlier, the effects of removing the logistical supply lines are very important considerations under study for the 1996 demonstration. By removing the enemies ability to re-supply you remove or severely impair his ability to fight.

When the base conditions for re-supply have been met, the MRL task spawns the company level service station and re-supply of the shooters is attempted. The re-supply behavior was implemented at both the company and platoon level. The company level task was developed from scratch while the platoon level task leveraged previous ModSAF development. At the company level multiple supply vehicles are used to facilitate re-supply under the following situations.

- The first situation is re-supply of an MRL company in the open. Each platoon in the company is tasked to re-supply from the closest, first available, undamaged supply vehicle. Each vehicle in the platoon receives supplies until the entire unit has been completely supplied. If there are more platoons than

supply vehicles, upon completion of a platoon re-supply, the waiting platoon would then be assigned to the available re-supply vehicle. If at any point during the execution of the re-supply task a supply vehicle is damaged the platoon terminates the current re-supply behavior and returns to its starting position. If any receiving vehicles are damaged while re-supplying then the next vehicle to receive supplies would begin re-supplying.

- The second situation is re-supply of an MRL company within a cave. The company level task determines the number of undamaged caves available, the number of platoons to receive supplies and how to distribute the platoons such that re-supplying them would be efficient. It then tasks the platoon(s) to travel to the cave to be supplied. Once the platoon enters the cave it is deemed concealed and the re-supply process begins. If the cave is damaged while the vehicles are re-supplying then the re-supply task terminates to reflect the damage to the cave, and any vehicles in the cave are considered trapped and unusable.

Upon completion of the re-supply mission, the MRL task will re-execute the same steps as described above, moving the unit to the next fire location and performing a new fire mission.

4. Future Work

Our future development will be guided by the needs of the JPSPD program and other CLCGF users. Possible future development areas include the following:

- integration with other fielded or prototype tactical equipment.
- continued support of both the Rapid Battlefield Visualization and the STOW Exercise Implementation efforts.
- enhancements to the existing JSTARS capability
- summarizing and reporting DIS indirect fire to Eagle so units played within the constructive model can be attrited by DIS indirect fire
- full parsing of Eagle OPORDs and the utilization of CCSIL to task disaggregated units.
- Work with LADS on the integration of a core aggregate capability into the ModSAF baseline.

5. Acknowledgment

This work is being sponsored by STRICOM, the Topographic Engineering Center (TEC), and the Depth and Simultaneous Attack Battle Lab at Ft. Sill, under the Joint Precision Strike Demonstration program, contract number DACA76-93-D-0007. We would also like to thank TRAC Ft. Leavenworth for their efforts in the development of Eagle scenarios, and the development of the Eagle <=> SIU socket interface.

6. References

Calder, R., Peacock Jr., J., Panagos, J., , Johnson, T., "Integration of Constructive, Virtual, and Engineering Simulations in the JPSD CLCGF", *Proceedings of the Fifth Conference on Computer Generated Forces and Behavioral Representation*, Orlando, FL, May 9- 11, 1995.

Calder, R., Peacock Jr., J., Wise, B., Stanzione, T., Chamberlain, F., Panagos, J. , "Implementation of a Dynamic Aggregation/Deaggregation Process in the JPSD CLCGF", *Proceedings of the Fifth Conference on Computer Generated Forces and Behavioral Representation*, Orlando, FL, May 9- 11, 1995.

Mitre Corporation, "Aggregate Level Simulation Protocol (ALSP) Program Status and History", March 1993.

7. Authors' Biographies

Jeffrey C. Peacock, Jr. is a Senior Software Engineer in the Technology Research Group at Science Applications International Corporation. He has been involved in the development of DIS CGF systems for past two years. Prior to entering the DIS CGF arena Mr. Peacock spent 7 years developing real-time embedded software systems. Mr. Peacock holds a Bachelors of Science degree in Computer Science from Merrimack College, Andover, MA.

Kevin C. Bombardier. is a Software Engineer in the Technology Research Group at Science Applications International Corporation. He has been involved in the development of DIS CGF systems for the past year. Mr. Bombardier holds a Bachelors of Science degree in Computer Science from Merrimack College, Andover, MA.

James Panagos is a Consultant for SAIC. He has been involved in the development of DIS CGF systems for over 10 years, and is currently performing design and development on the ARPA Command Forces (CFOR) project. His primary research interests are in the area of tactics, behavior representation, automated planning and generation for computer generated forces. Mr. Panagos has a Master

of Science degree in Computer Science from Massachusetts Institute of Technology.

Thomas E. Johnson is a Senior Software Engineer in the Electronic Systems Division at Raytheon Company. He has been involved in the development of real-time simulation models and manned flight simulators over the past fourteen years. His primary interests are in the area of man-machine interface and the development of tactical simulation models. Mr. Johnson has a Bachelor of Science degree in Aerospace Engineering from the Virginia Polytechnic Institute and State University, Blacksburg, VA.



A Strategic Plan For The Integration Of ModSAF And CCTT SAF

Major John D. Norwood, Ph.D.
Simulation, Training, and Instrumentation Command
12350 Research Parkway, Orlando, FL 32826-3276
norwoodj@stricom.army.mil

1. Abstract

In recent months, there has been increasing interest in the potential for integrating the functionality of two separate Semi-Automated Forces (SAF) products -- Modular Semi-Automated Forces and Close Combat Tactical Trainer SAF (CCTT SAF). The overarching concept is to develop and test an underlying SAF architecture that can be used to combine or *integrate* the features and functionality of these products. This concept originally grew out of meetings between the Project Manager for Combined Arms Tactical Trainer (PM CATT) and the Defense Advanced Research Projects Agency (DARPA), as a concept for technology transfer. SAF Integration has been subsequently incorporated into the Computer Generated Forces (CGF) Assessment conducted by the Army Materiel Systems Analysis Activity (AMSAA). Additionally, both the Deputy Under Secretary of the Army for Operations Research (DUSA(OR)) and the Army Acquisition Executive have endorsed the program as the Army strategy for its potential in eliminating duplication of effort in SAF development and potentially facilitating the migration to a single SAF product. This paper presents an overall strategy consisting of a series of experiments and technical assessments which address key issues in achieving an integrated SAF for the Army.

2. Background

Semi-Automated Forces (SAF) is an area of considerable interest to the modeling and simulation (M&S) community. SAF allows one to represent multiple, unmanned entities in the synthetic environment with some measure of intelligent behavior and human-like functionality. SAF first began as an outgrowth of the Defense Advanced Research Projects Agency (DARPA) Simulation Networking (SIMNET) program. Early on, it was understood that the cost of fielding large numbers of manned

simulators was prohibitive, and that a method of *fleshing out* the forces with tactically meaningful numbers of entities must be found. A second motivation for developing SAF was that an intelligent enemy force was needed for training in the synthetic environment. The eventual solution was a SAF system, in which multiple entities were controlled from a single, low-cost computer platform. While SIMNET was well received in the training community, it was fielded with limited functionality and documentation and was not easily modifiable to include new or enhanced capabilities.

ModSAF (Modular Semi-Automated Forces) traces its lineage to the SIMNET program. As its name implies, ModSAF modularized the software code associated with both SIMNET SAF and ODIN (73 Easting) SAF. In 1993, DARPA began building ModSAF by developing an open architecture, which could be used to create synthetic agents for a variety of Distributed Interactive Simulation (DIS) applications. The initial effort fielded a system in December 1993 to support the What-if Simulation System for Advanced Research and Development (WISSARD) program, which had a requirement for beyond-visual-range, air-to-air engagement scenarios. After the initial release, additionally Battlefield Operating Systems (BOS) and behaviors were added to ModSAF in order to fill out the synthetic battlefield. Concurrently, the Simulation, Training, and Instrumentation Command (STRICOM) agreed to fund an effort to document this revised code. ModSAF 1.2 was released in June 1994 and included the majority of systems that had been represented in the previous SIMNET SAF version.

Close Combat Tactical Trainer (CCTT) SAF is being developed and fielded as an integral part of the CCTT program. When CCTT was awarded in November 1992, SAF development was considered the highest risk area of the CCTT

program, potentially having significant performance, schedule and cost risks. Initially, the CCTT program focused on developing a new SAF product; however, after identifying several major design deficiencies with the chosen design approach, a SAF Trade Study was initiated to determine the best way to proceed. The Trade Study recommended that ModSAF be re-engineered to accommodate the CCTT software environment, while incorporating many new functional, system, and visual requirements specific to CCTT. To facilitate this re-engineering effort, CCTT developers were placed at the ModSAF development facility in Cambridge, Massachusetts, and a ModSAF engineer was hired for CCTT. Thus, while CCTT SAF was an evolution of the ModSAF baseline; significant changes were made which caused it to diverge from the ModSAF 1.2 baseline. This divergence hinders compatibility, data interchange, and software reuse between the two products.

While ModSAF and CCTT SAF evolved from a similar backgrounds, they are in fact very different products. Key reasons for this disparity are that they were developed for different simulation domains, are intended for use by different simulation communities, and have differing needs in terms of differing verification, validation, and accreditation (VV&A). ModSAF, built by DARPA has been used primarily as a research tool and has found wide-spread support in the research and development community. On the other hand, CCTT SAF is focused primarily on training issues and must be verified, validated, and accredited (VV&A'ed) by the training community. This situation springs from the necessary dichotomy in modes of operation between DARPA, a technology developer, and a program management office, like PM CATT. DARPA develops technology with sufficient representative functionality to demonstrates the utility of the approach to an operational user. Conversely, a PM office focuses on providing the required functionality with sufficient technology to meet a user's set of approved operational requirements. Key to this effort is a demonstrated commitment between DARPA and PM CATT to transition technology as it is developed to provide the operational user with increased capability and functionality. This commitment is most evident in the Command Forces Memorandum of Agreement, which

stresses a cooperative effort in development and transition of CCSIL. A similar agreement is appropriate for SAF Integration.

In January 1995, Mr. Walter Hollis, Deputy Under Secretary of the Army for Operations Research, commissioned the Army Materiel Systems Analysis Activity (AMSAA) to assess the status of the major SAF systems currently in use throughout the Army. The specific purpose of the CGF Assessment was to evaluate alternative CGFs for all Army DIS Domains: Advanced Concepts and Requirements (ACR); Training, Exercises, and Military Operations (TEMO); and Research, Development, and Acquisition (RDA). A key conclusion of the CGF Assessment was that no CGF assessed meets or will meet all M&S domain application requirements. This point is also clear from the discussion presented above. SAF design decisions are driven by the requirements of the simulation domain for which they are built. A second and complementary conclusion is that while pursuing one CGF to meet the needs of the entire community is conceptually feasible, it may not make optimal use of existing resources and investments. In fact, the leap from conceptualization to implementation may be daunting in terms of the technology and investment required to be all things to all users. Finally, managing the development of a single SAF system is highly dependent upon the ability to bring together the right people and funds in a management structure that can both develop the system and meet the requirements and milestones of the user community.

This document formally describes the SAF Integration program and presents an overall strategy for executing the program.

3. Program Goals and Objectives

The overall goal of the SAF Integration program is to develop and prototype an objective SAF architecture, which integrates or combines the functionality of both ModSAF and CCTT SAF. To accomplish this goal, it is necessary to identify and assess several intermediate objectives.

- Establish a greater degree of interoperability between ModSAF and CCTT SAF to facilitate experimentation and evaluation of technical issues.

- Identify and assess the technical barriers/challenges to integrating ModSAF and CCTT SAF.
- Crosswalk the synthetic environment terrain representation requirements for ModSAF and CCTT SAF and integrate these with the DARPA Integrated CGF Terrain Data Base (ICTDB) program.
- Propose a flexible and extensible SAF arch-

- Build community consensus for migrating to the objective system.

4. Experiments

The SAF Integration program is built around a series of experiments, which progressively assess key issues in interoperability and integration of capability. These experiments are shown graphically in Figure 1. The major issues affecting SAF integration are: synthetic

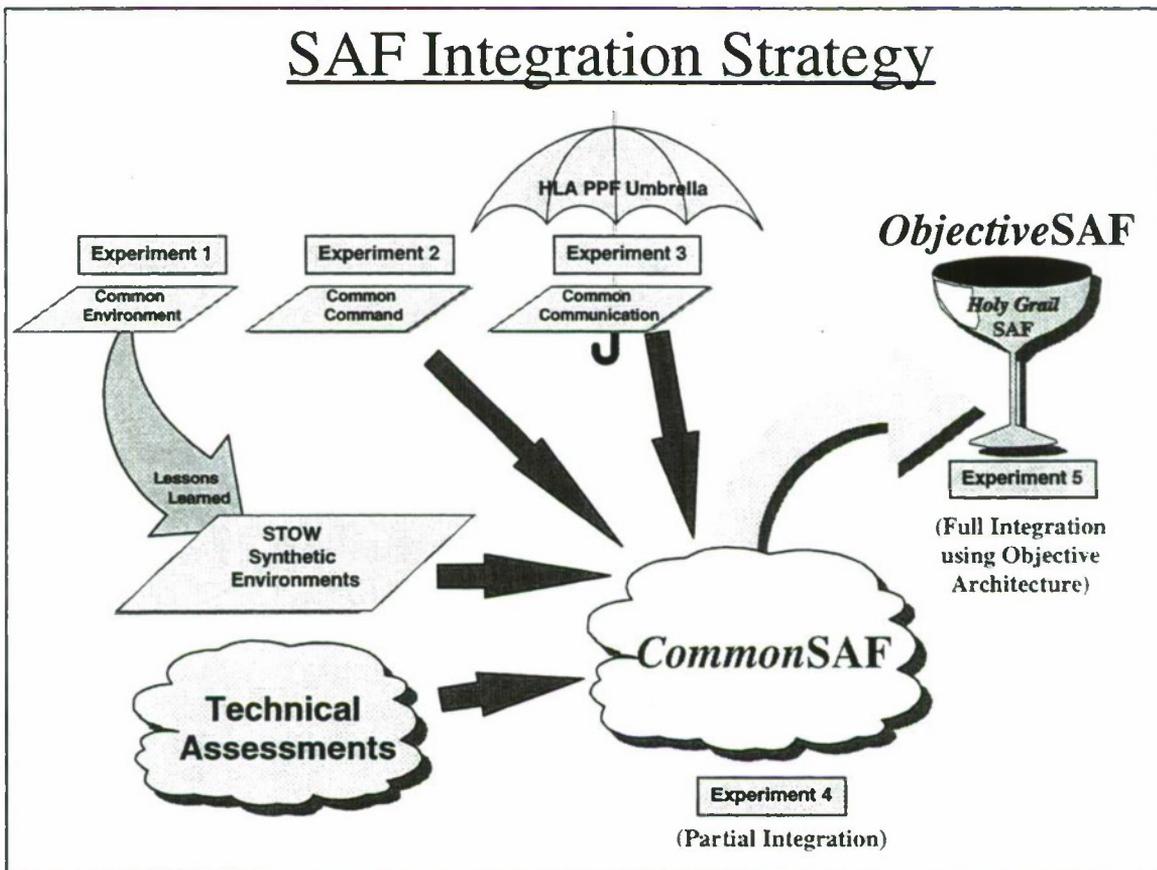


Figure 1: SAF Integration Strategy

- itecture for integrating ModSAF and CCTT SAF, which considers (and incorporates, as appropriate) the on-going developments in the DMSO High Level Architecture, Distributed Interactive Simulation, and the Synthetic Theater of War (STOW) 97 programs.

environment interoperability, command and control, distributed network communications, and SAF architectural design. The experiments presented below attack these issues and draw conclusions, which become important in assessing and implementing the re-architecture of the two systems.

4.1. Synthetic Environment Interoperability

This experiment addresses the idea of a common synthetic environment representation for ModSAF and CCTT SAF. Currently, ModSAF uses a terrain representation known as Compact Terrain Data Base (CTDB) and CCTT SAF the Model Reference Terrain Data Base (MRTDB). These terrain databases differ significantly in their capability. To increase compatibility, CTDB should be enhanced to represent new MRTDB features, including: additional soils types, forest templates or *basis sets*, variable grid diagonalization, etc. In fact, recent releases of ModSAF (versions 2.0 and 2.1) incorporated many of these improvements into the CTDB format 4. The focus of this first experiment is to evaluate selected terrain functionality into a CTDB format 5 and assess performance and capability tradeoffs and provide this feedback into both CCTT and the ICTDB program. This will provide a measure of interoperability to the two SAF products and provide a basis for future enhancements and technology transfer between systems.

4.2. Higher Level Command and Control

This experiment builds on the first experiment by assessing the ability of a command entity to simultaneously control both ModSAF and CCTT SAF in the execution of a common mission. The command entity will be represented by leveraging the Command Forces (CFOR) program and the Command and Control Simulation Interface Language (CCSIL) being developed for DARPA by the Mitre Corporation. This experiment requires that both ModSAF and CCTT SAF implement CCSIL. In fact, a ModSAF CCSIL interface has already been developed and is being extended by the STOW 97 program. This experiment proposes to integrate CCSIL with CCTT SAF and to communicate commands to both several ModSAF and CCTT SAF units from a higher command entity hosted on either system. A secondary goal is to integrate this work with the High Level Architecture (HLA) and the runtime Infrastructure (RTI). This experiment accomplishes several objectives. First, it exercises the CCSIL interface by using it to control entities separate of ModSAF -- its development environment. Secondly, it provides valuable insight into the issues associated with command and control of the two independent SAF systems. Third, it demonstrates an ability to

task organize using different simulation assets within an overall simulation exercise.

4.3. High Level Architecture

This experiment identifies and assesses issues associated with ModSAF and CCTT SAF interoperating through the HLA RTI. Since CCTT SAF is participating in the HLA Platform Proto-Federation (PPF), this experiment leverages the already planned (and funded) HLA PPF demonstration for integration of the RTI into CCTT SAF. Additionally, the STOW 97 program is integrating the RTI into ModSAF. Both systems will be HLA compliant and available for experimentation by late summer 1996. Experiments in the PPF and between the SAF systems will stress the RTI and provide valuable feedback to DMSO for improving and streamlining the RTI for entity-level, real-time simulation systems. It should be noted that neither SAF is fully integrating the HLA RTI fully within the systems. This experiment will focus on providing data as to the long-term utility of this integration effort.

4.4. Common Services

This experiment uses the results of the three previous experiments, the output of the Technical Assessments (described below), and ongoing work and products developed in STOW 97 to specify, develop, and implement a prototype objective architecture for SAF. The DARPA STOW program has recently been designated as the developer of simulation technology for the Joint Simulation (JSIMS) program. In this role, it is anticipated that a new simulation infrastructure will be developed suitable for entity- and aggregate-level simulation. These plans are currently under development and will be finalized in July 1996. It is anticipated that this infrastructure will provide common simulation services and be applied across the services in next generation simulation systems. The Army must maintain awareness of these developments and evaluate their utility to entity-based, platform-level simulation systems, like ModSAF and CCTT SAF and assess their applicability in integrating the two products. This experiment will give insight into the viability of the new architectures for an objective SAF and serve as a decision point for continuing the SAF Integration program.

4.5. Objective SAF Architecture

The final experiment builds upon all previous experiments by further decomposing ModSAF and CCTT SAF on an objective SAF architecture. The primary objective of this experiment is to determine the level to which both SAFs can be decomposed and to what extent the component parts can be shared. This experiment addresses how various component (models, algorithms, behaviors, etc.) can be shared as common services, to what extent they can be interchanged for a given application, and what composition relationships must exist in configuring a SAF application. This experiment pushes the state-of-the-art in architectural developments and will provide insight into entity level simulations in a global sense. This experiment leverages STOW architectural development and improves upon it where appropriate. The results and conclusion may be useful to such ongoing programs as CATT, WARSIM, and JSIMS.

5. Technical Assessments

A key premise of the SAF Integration program is that much of the ModSAF and CCTT SAF functionality can be shared and/or migrated from one system to the other. It is important to evaluate many key technical areas to assess implementation issues, degrees of software reuse, impacts of new technologies, and the ability to migrate functionality to an objective SAF system. These assessments will generate insight into the issues associated with developing an objective SAF architecture and reusing components and features of both ModSAF and CCTT. The following technical assessments have been identified to date:

5.1. Software Languages

Assess the ability of Ada 95 and Common Object Request Broker Architecture (CORBA) to overcome computer software language differences between ModSAF and CCTT SAF. Both Ada 95 and CORBA have the potential to incorporate existing software without a significant recoding effort.

5.2 Reuse

Assess the implementation issues associated with incorporating the CCTT SAF Hulls Computer Software Component (CSC) into ModSAF. The Hulls CSC contains the CCTT SAF mobility codes, which provide enhanced functionality to ModSAF. This assessment provides insight into the ease of software reuse and the ability of the ModSAF Generic Model Interface (GMI) to accept CCTT SAF code.

5.3 Global Coordinate System (GCS)

Assess the impact of the Global Coordinate System (GCS) on CCTT SAF and the objective SAF architecture. GCS provides a significant enhancement over coordinate projection systems currently used in SAF. GCS provides curvature of the earth information to local coordinate systems. A key issue will be the system impacts of implementing GCS on both SAF Systems.

5.4 Behaviors

Assess implementation issues associated with migrating and/or sharing entity behaviors between ModSAF and CCTT SAF. CCTT SAF has developed a library of verified, validated, and accredited (VV&A'ed) behaviors using a methodology of developing and implementing Combat Instruction Sets (CIS). Additionally, many ModSAF behaviors have been VV&A'ed for the Anti-Armor Advanced Technology Demonstration (A2ATD) and other programs. Since both systems use a Finite State Machine (FSM) architecture for implementing behaviors, it seems plausible that behaviors could be migrated from one system to the other and certainly to the objective system.

5.5 Private Protocols

Assess the ModSAF Persistent Object Protocol (POP) and the CCTT SAF Synthetic Environment Object Database (SEOD) for commonality and applicability to an objective SAF architecture. Currently, both ModSAF and CCTT SAF provide command and control communications using private protocols. It is important to understand the POP and SEOD implementations in structuring the objective architecture.

5.6 Synthetic Environments

Assess the DARPA Synthetic Environments (SE) work currently on-going in Dynamic Virtual Worlds (DVW), Weather in DIS/Total Atmosphere and Ocean Server (WINDS/TAOS), and Dynamic Terrain and Objects (DTO) for implementation in CCTT SAF and in the objective architecture. STOW SE is developing and integrating significant SE enhancements to ModSAF, these improvements should be evaluated for applicability and use in the integrated SAF system.

5.7 Terrain Format

Assess the ICTDB representation for implementation in the integrated SAF. ICTDB is already supplying a synthetic environment for ModSAF. An important part of this assessment will be to provide a complete set of requirements to support both SAFs. Any gaps or missing functionality must be evaluated with respect to its future impact on both systems. Also, key issues, such as the use of integrated triangulated irregular network (ITIN) must be assessed.

6. Key Issues

Several key issues have been identified that impact on an eventual SAF integration. These issues must be resolved over time and determine to a large extent the form of the objective SAF architecture.

6.1 Monolithic System versus Software Repository

The concept of a single SAF system may be outmoded. One can envision a SAF that serves multiple users and applications by linking together software modules retrieved from a central software repository. However, this concept presupposes a flexible and highly extensible SAF architecture that can dynamically link diverse software components to form a SAF construct for a specific application.

6.2 Common Services

There is much discussion currently concerning the idea of providing common services via a distributed architecture. An important product of this effort will be to identify these common services and confirm that they can indeed be

distributed in some fashion in a real application. Identification of these services will impact directly on development of the objective architecture. This issue is being addressed partially by the HLA.

6.3 Common Terrain

The DARPA ICTDB effort is being incorporated directly into ModSAF. It is important to understand the structure and implementation of ICTDB and ensure that future development incorporates the requirements of CCTT SAF. To better understand this issue a capability matrix must be developed which crosswalks the capabilities of ICTDB, MRTDB, and CTDB. This matrix will highlight commonality and identify gaps and shortfalls with respect to the various formats.

7. Program Management

7.1 Participating Agencies

Several DoD programs are developing technology which is crucial to the SAF Integration program, several of whom are identified above. Additionally, this program has received high level attention throughout the DoD. While SAF Integration will be principally managed in PM CATT, an Integrated Product Team (IPT) will also be formed to guide the overall development effort. The IPT will initially consist of participants from the following agencies: STRICOM, AMSAA, the National Simulation Center, the TRADOC Analysis Agency, and DARPA. The IPT will be chartered and chaired by PM CATT.

7.2 Related Programs

Several programs are currently underway, which potentially have interest in the interoperability and eventual integration on ModSAF and CCTT SAF. These programs may be leveraged directly by focusing experiments and/or technical assessments to produce results of particular interest. However, the concept of leveraging for SAF Integration, while necessary to the success of the effort, does not appear to be sufficient to ensure overall program success. Programs with direct interest in SAF Integration include but are not limited to: STOW 97, JSIMS, WARSIM,

ModSAF developers, CCTT, and the High Level Architecture.

8. References

DMSO Survey of Semi-Automated Forces, July 30, 1993.

A Self-Assessment of CCTT SAF, March 1, 1995.
Modular Semi-Automated Forces (ModSAF) Review by the Computer Generated Forces (CGF) Evaluation Team, June 30, 1995.

Minutes of SAF Merge Meeting, June 25, 1995.

Minutes of SAF Merge Meeting, July 20, 1995.

Minutes of SAF Merge In-Process Review, Sept. 19, 1995.

Memorandum of Agreement between DARPA and PM CATT on Interoperability of DARPA Command Forces and STRICOM CCTT Semi-Automated Forces (CCTT-SAF), 2 June 1994.

9. Biography

Major John D. Norwood is assigned to the Simulation, Training, and Instrumentation Command (STRICOM) as an Assistant Project Manager for the Combined Arms Tactical Trainer program. He holds Master of Science and Doctor of Philosophy degrees in Mechanical Engineering from Rice University, concentrating in the field of robotic control. He has been selected for promotion to the rank of Lieutenant Colonel.



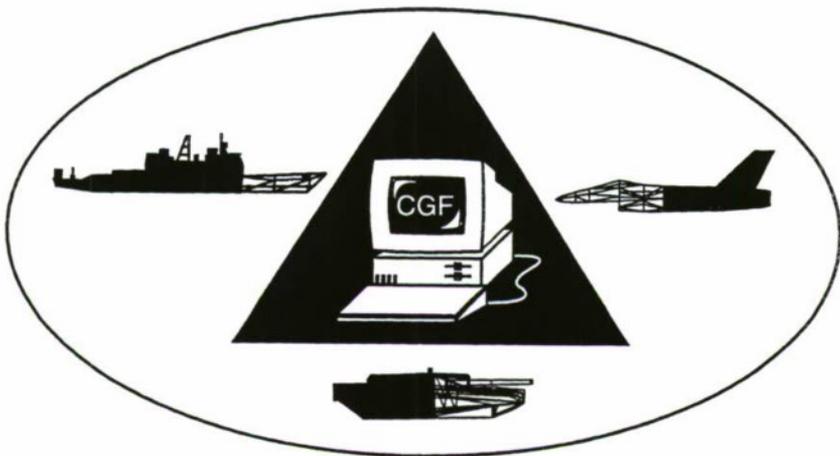
Session 5a: Agent Architecture

Adkins, U. S. Army, TRAC, Ft. Leavenworth

Hepplewhite, DRA, UK

Kenny, University of Michigan, AI Lab

Reece, UCF/IST



Polling vs. Event-driven Computer Generated Forces (CGF) Architectures

Michael K. Adkins
TRAC-OAC-MRD
Attn: ATRC-FM Mr Adkins
255 SEDGEWICK AVE
FT LEAVENWORTH KS 66027-2345
adkinsm@trac.army.mil

1. Abstract

This paper presents a study using a control flow model of the ModSAF scheduler. The model consists of two parts: a representation of a polling architecture like that used in ModSAF and one that accomplishes the same tasks under an event-driven architecture. The purpose of this study was to compare the two approaches as they relate to a CGF system and analyze the results to see which, if either, is more efficient. The main focus was on the scheduler, because it is an important efficiency area in CGF systems.

The experimental part of the study included collecting data from runs of ModSAF. The number of ticks that were recorded for one specific function in the loop was used for input for the two experimental models. Results show the differences in efficiency of the two architectures.

2. Introduction

CGF systems depend on the management of many simultaneous tasks. This is accomplished using scheduling methods. The two most common methods are *polling* and *event-driven*.

In a polling architecture, each task in the system is processed either at regular time intervals or each time the system makes its way through its list of tasks. The tasks are "checked on" at these intervals in order to see if they have a current need for processing. If there *is* something to do, then control is given to the task, otherwise, the next task in the list is invoked.

An event driven architecture is one in which each task in the system receives processing time upon the occurrence of an event. The system does not spend any time with a task until something occurs which indicates that it requires processing. This ensures that a task is not invoked unless it needs to be.

As simulation technology advances, users expect to run larger applications with higher fidelity. Demanding simulations impede performance of even the fastest systems. Small improvements in efficiency can return significant gains in performance. In a CGF system, increasing demands due to the

growing size and complexity of simulations require efficient software designs to ensure maximum performance. Some implementation methods are more efficient than others.

In a CGF system, function calls are used to achieve a specific goal through transitions from one state to another. For example, the goal of a vehicle may be to move to a location. It might begin by transitioning from a state of "stationary" to a state of "moving in the direction of the desired location". Other state changes occur as necessary until the goal is achieved.

ModSAF handles this process by polling or "ticking" function calls at specified time intervals. This is accomplished by placing them into a "ring". Each process, in this case function call, in the ring is periodically "ticked" to see whether it requires processing time. If it does, then the appropriate action is taken. Otherwise control is bypassed to the next process. It is possible that an event-driven process would be more efficient, since time would not be spent checking to see if a specific function needs to be called. Instead, it would be given control only when needed.

One purpose for this project was to compare the two scheduling architectures as they relate to a CGF system and to analyze the results to see which method, if either, is more efficient. Another reason was to determine a method for measuring inefficiencies related to scheduling. ModSAF was the CGF system used in this comparison.

3. Scheduler

The ModSAF scheduler orchestrates the entire simulation process by scheduling function calls based on time priorities and then invoking those functions as timely as possible in an attempt to keep all entities and tasks updated throughout the course of the simulation.

3.1 Relevant Definitions

ring - A ring is a data structure used by the scheduler to keep track of functions that need to be invoked at a specific time interval. ModSAF creates

two rings by default. One ring contains functions that need to be invoked every 67 milliseconds and the other calls them every 29000 milliseconds. More rings can be added for different time intervals.

tick - A function that gets called periodically is referred to as having been “ticked” every time it is invoked by the polling mechanism (i.e. scheduler), so each call to a function is referred to as a tick.

3.2 Overview

In a CGF system, entities achieve specific goals through a series of transitions from one state to another. One method of accomplishing this is by calling specific functions which perform the steps required for the desired change to take place. An example would be a vehicle with a goal of moving from one location to another. If the vehicle were sitting still and then set into motion toward its intended destination, it could be seen as having changed from a state of “stationary” to one of “moving in the direction of the desired location”. The appropriate function or sequence of functions would be called to accomplish this move.

ModSAF handles this process by polling or “ticking” function calls at specified time intervals. Each function is first placed onto a priority queue. Its position (i.e. priority) in the queue is determined by the time in which it is scheduled to execute. If a function is to be called only once, then it is removed from the queue, executed, and then removed from the system. If it is to be called periodically, then it is removed from the queue, executed, and then scheduled onto a ring with other functions that need to be invoked at that same time interval. Each function call in a ring is then periodically “ticked” or invoked to see if it needs to update the status of a specific entity or process. If updating is required, then the appropriate action is taken. Otherwise the next function in line is invoked to see if it has work to do.

Since the functions have to check to see if there is work to be done, it is possible that an event-driven process would be more efficient, since time would not be used checking for work. Instead, functions would be invoked only when they need to be.

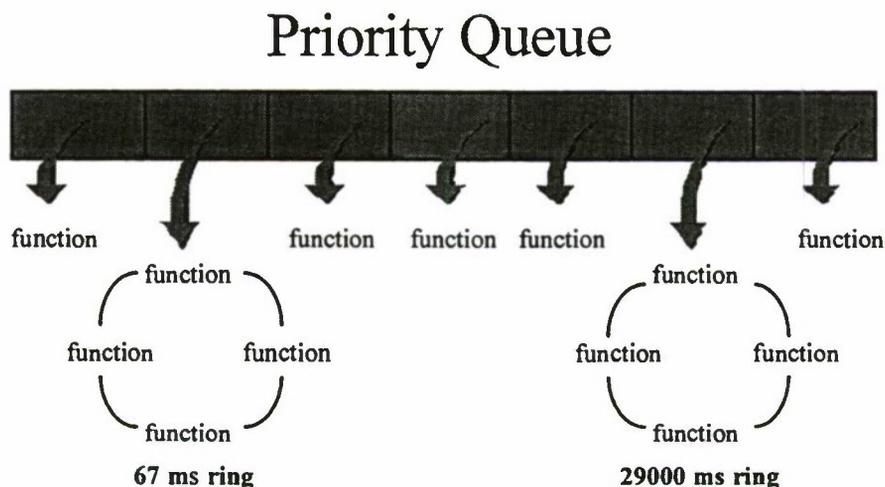


Figure 1 : Basic Structure of the ModSAF scheduler

4. Project Description

The first step was to study the ModSAF scheduler in order to learn which scheduling method it employs (polling or event driven). As stated earlier, the scheduler uses a polling architecture.

Two experimental models representing the polling and event driven architectures were developed. The

number of productive and non-productive function calls were recorded from a run of ModSAF and used to formulate input. In order to keep the amount of time spent on productive function calls the same, a delay function is called using the same amount of delay (`delay_time`) in both models. The polling program also allows the user to dictate the maximum number of passes (`max_passes`) that can be made through the “tick loop”, the number of processes

(number_of_processes) in the tick loop and the ratio of good to bad ticks, which is controlled by the (process_interval). If the process_interval = n, then every "nth tick" would be counted as a good tick.

The models helped to discover some differences between the two architectures. One of the most noticeable differences is that a polling architecture keeps querying the system to see if there is work, while an event driven one waits for the appropriate event to occur. When there are only a few tasks in the system, the polling method may out perform the event driven, but when the number of tasks becomes large enough, the overhead in polling may be enough to allow the performance of the event driven system to prevail. A variety of different numbers of tasks were used as input for the two models. Results support polling as the quicker method with a small number of tasks, but the event driven surpasses it once the number of tasks grows large enough.

5. Models

The polling and event driven models represent straightforward implementations which simulate the bare essence of each architecture, in order to compare the differences in efficiency of the two methods.

5.1 Input Data

A function call is considered to be productive if it performs or invokes a task which is needed by the simulation at the time the call is made. If a function call is made when there is no task to be performed, then it is considered non-productive.

For example, if a run resulted in a case where one-half of a total of 1000 "functions ticked" were during times in which the ticked process had something to do, then those 500 ticks would be considered productive, leaving the remaining 500 as non-productive.

The number of productive and non-productive ticks were recorded from runs of ModSAF and used to formulate input for the two models.

The ModSAF library Libtracked handles the changing of states of all tracked vehicles running in a ModSAF scenario. This library was selected as the one to gather data from because all that had to be done to ensure that it was placed into the scheduler and repeatedly invoked was to create tracked vehicles. The file `trk_tick.c` was modified to count the number of ticks in which something productive was accomplished and the number in which there was no reason to have made the call. The amount of time which elapsed in each of the instances of non-

productivity was recorded and the average time spent in each was calculated.

5.1.1 Input for the polling model

The following four variables were used as input for the polling model.

`delay_time` = The amount of time in milliseconds to delay on a productive tick. This time is always the same for all ticks. This helps ensure that the amount of productive time spent processing can be easily accounted for and duplicated in both models.

`max_passes` = The total number of times to pass through the tick loop.

`number_of_processes` = The total number of processes in the tick loop.

The product of (`max_passes` * `number_of_processes`) must be equal to the total number of ticks from a ModSAF run. This gives the option of ticking a lot of processes a few times or ticking a few processes a lot of times, while still having the desired number of total ticks.

`process_interval` = $\frac{\text{number_of_total_ticks}}{\text{number_of_good_ticks}}$. This number allows the user to dictate the number of good and bad ticks. For example, if we let (`n` = `process_interval`), then we can count every "nth tick" as a good tick.

5.1.2 Input for the event driven model

The following two variables were used as input for the event driven model.

`delay_time` = The amount of time in milliseconds to delay on the occurrence of each event. In an event driven system, all events are productive, so the delay is called for each one.

`number_of_events` = The number of good ticks from a ModSAF run.

5.2 Implementation

In the example from section 4.1 above, the polling program would cycle 1000 times, invoking a delay function every time there was supposed to be something productive done by the task or process. To determine when a tick is supposed to be productive, the total number of ticks is divided by the number of productive ticks. The result (`n`), is how often a tick should be productive (i.e. every nth tick). During the non-productive cycles, the program

would just pass through, not calling the delay function. It is the overhead of passing through without doing any productive work that is being questioned as far as its effect on the overall efficiency of the process.

The event driven program would invoke the total number of productive ticks, 500 in this case, taking one right after the other from a queue and calling the delay function for the same amount of “delay time” as in the polling process. The difference is that there wouldn’t be the overhead of passing through without doing any work that was experienced with polling. It is important that the amount of productive processing time spent in both models is the same, so that the only time difference is in what happens during times when no productive processing occurs. It is the amount of time spent on non-productive activities such as polling that must be minimized so that more time is available to processes that need it.

The overhead in polling as opposed to the lack of it in an event driven system is what prompts the question of differences in efficiency between the two methods.

6. Results

6.1 ModSAF

The results of the ModSAF efficiency portion of this study are based on information gathered from the ticking of the tracked_tick function in Libtracked. This is the only function that was observed in determining the number of productive and non-productive function calls made by the scheduler, as well as in calculating the amount of time used on non-productive function calls.

If productive work was done somewhere in the function, a flag was set to true to denote a “good tick”. Otherwise, the flag would be returned from the function with a false value and be considered a non-productive tick.

Any function that is placed into the tick loop process of the scheduler could be used, as well. In fact, this information would have to be collected from *all* of the functions ticked during a run to obtain accurate results from the entire simulation.

The average time spent on non-productive ticks ranged between .02 and .05 milliseconds for each tick. If most of the tracked entities were moving or engaged in some type of activity, then the number of non-productive ticks and wasted time was minimal. As the entities stopped because their task had ended or due to damage from enemy fire, the number of non-productive ticks and the amount of time wasted would rise. The amount of time spent and the

number of non-productive ticks is scenario dependent due to the fact that there are usually different numbers of entities engaged in different types of activities. Different runs of the same scenario can also result in different outcomes due to the randomness of some elements, therefore the percentages of productive and non-productive ticks and time wasted will vary here as well.

If there are a lot of vehicles destroyed early in a run, then the number of bad ticks accumulated due to ticking dead vehicles will be higher than on a run when most vehicles survived until near the end. The reason for this is because the tracked_tick function is still ticked, even when the tracked vehicle it is checking has been destroyed. This means that as the number of destroyed tracked vehicles increases, so will the amount of non-productive time spent ticking them. In a scenario with multiple engagement areas, once the tasks are finished or a lot of kills occur in one area, then more time will be spent ticking these vehicles instead of being used elsewhere in the scenario.

The same can be said for scenarios in which entities are halted in one position while waiting for a specific event or period of time to elapse before moving or executing a task. During the waiting period, very little productive work occurs in the tracked_tick function.

In order to compensate for the variability of different scenarios, a series of scenarios could be created, with each one representing different levels of activity. The resulting data would then be compiled together to find an average to represent the entire variability spectrum.

6.2 Models

The amount of time it takes for a model to perform all of its ticks (polling) or invoke all of its events (event driven) was used as the measure of efficiency. The total time it took for the polling model to perform the designated number of “productive and non-productive ticks” was compared to the amount of time it took for the event driven model to invoke a number of events equal to the number of productive ticks performed by the polling model. Each time a model encountered a “productive tick” or an “event”, a delay function was called so that each tick or event lasted the same amount of time in both models. Therefore the only difference in the amount of elapsed time should be the overhead required by the nature of the architectures themselves plus the time spent on non-productive ticks in the polling model.

In each instance of testing with actual data from a ModSAF run, the polling and event driven models gave similar results for the amount of time each one required to complete the same percentage of tasks, with each task completed taking the same amount of time. Even when the number of non-productive ticks was extremely high, the overhead in polling them was not enough to make the event driven system noticeably more efficient.

A comparison of the two models was performed with a series of tests. The process interval was set at 20 for each test. This means that in all of the tests, only 5% of the total number of ticks were productive ticks

in which the delay function was called. These numbers would be a rare occurrence in actual runs, but having a situation bad enough that its occurrence would be uncommon in actual ModSAF runs would be a good measure for a "worst case" scenario to judge results against.

Polling proved to be the faster method when only a few tasks were used. As the number of tasks increased, the event driven became faster.

The following table shows the results from the different numbers of tasks used as input.

Polling		Event Driven	
Number of Tasks	Elapsed Time	Number of Tasks	Elapsed Time
500	2000	25	2,500
1000	5000	50	5000
25,000	125,200	1,250	125,000
50,000	250,400	2,500	250,200
100,000	500,900	5,000	500,200
1,000,000	5,006,900	50,000	5,003,800

Table 1 : Polling and Event Driven Comparisons

A ModSAF scenario consisting of only tracked vehicles was run for 100,000 ticks of the tracked_tick function. The number of productive ticks in this instance was approximately 50%. Representing this data as input for the polling model consisted of 100,000 total tasks, with a process interval of 2. This allows every other task invoked to be a "productive tick". The event driven model had to process 50,000 events, which represents 50% of the total number of tasks used as input to the polling model. The results showed that the event driven model completed all of the tasks in 83.414 minutes, which was approximately 1.6 seconds faster than the 83.440 minute time period used by the polling method.

7. Conclusions

Results suggest that there are areas in CGF systems where an increase in efficiency could be beneficial. In particular, during times of lesser activity, a significant amount of processing time can be spent on non-productive polling in the updating of tracked vehicles. To provide a more accurate result, other areas handled by the scheduler could be checked using the same method.

Results from the comparison of the polling and event driven architectures do not lead one to believe that one method would be significantly better than the

other with regard to the ModSAF scheduler's handling of tracked vehicles. However, that does not necessarily mean that the same conclusion would be drawn in other areas of ModSAF. All areas that are handled by the scheduler would have to be tested to lead to that conclusion.

This project provided valuable insight into simulation architectures and some of the factors that influence their performance. Further investigation would provide more in-depth information and possibly lead to other conclusions.

8. Future Work

The methods described in this paper could be used by others to perform similar tests in other areas of ModSAF, or in other simulation models.

The level of sophistication of the experimental polling model could be increased by varying the duration, frequency and types of tasks used in the model. These changes would provide a more accurate representation of the polling architecture used by the ModSAF scheduler.

The mechanism used to time the functions and models returned results in milliseconds. It is possible that the execution time of some functions may be less than one millisecond. Therefore,

fractions of time that need to be recorded are being lost. Getting the results in microseconds may provide more accurate results.

9. Acknowledgment

The author would like to thank those individuals at the Institute for Simulation and Training for their support and assistance in this project which began while at IST for technical training as part of the TRAC internship development program. Thanks is also given to Mr. Kent Pickett of the TRADOC Analysis Center at Fort Leavenworth who provided editorial assistance.

10. References

- Dumpleton, G. "Event Driven Systems", *OSE-C++ Library User Guide*, Dumpleton Software Consulting, Pty Limited, N.S.W. Australia.
- Evans, J.B. (1988). *Structures of Discreet Event Simulation*, Halstead Press: a division of John Wiley & Sons, 1988, pp 77-101.
- Johnson, M. "Interrupts vs. Polling", *Linux Kernel Hackers Guide Version 0.5*, Linux Documentation Project, subsection 2.3.3.4.
- Levy, H. (1990). "Polling Systems: Applications, Modeling, and Optimization", *IEEE Transactions on Communications*, vol. 38, No. 10, October 1990, pp. 1750.
- Silberschatz, A. and Galvin P.B. (1994). *Operating Systems Concepts*, Addison-Westley Publishing Company, Inc., June 1994.

11. Author's Biography

Michael K. Adkins is a Computer Scientist for the Modeling and Research Directorate of the TRADOC Analysis Center at Fort Leavenworth, Kansas. He has earned a Bachelor of Science degree in Computer Science and in Mathematics. He is currently pursuing a Master of Science degree in Operations Research. His research interests are in the areas of simulation, distributed computing and artificial intelligence.

Broad Agents for Intelligent Simulation

R. T. Hepplewhite and J. W. Baxter

Defence Research Agency,
St. Andrews Road, Malvern, Worcs, UK.
EMAIL {rth,jbaxter}@signal.dra.hmg.gb

1. Abstract

Computer Generated Forces (CGF) are becoming increasingly important for controlling entities within Synthetic Battlefields. The thrust of CGF development is reducing the large number of operators required to control battlefield units when training only a few military commanders.

These automated forces must operate in a spatially and temporally continuous domain, react to any situation in a realistic but non-deterministic manner, using potentially uncertain information about the world.

This paper describes a "broad agent" approach to implementing intelligent entities for battlefield CGF systems, which attempts to avoid using detailed rules covering every possible contingency but instead uses more fundamental principles for deriving plans to achieve objectives. These agents are linked in a command and control (C²) hierarchy. The agent's behaviour is implemented as rule sets executed by a tool-kit, developed to support generic agent architectures. The tool-kit supports multiple objects and mechanisms enabling the agents to interact in synthetic environments.

When the command agent receives an objective it generates a plan by considering possible sequences of actions selecting the most appropriate plan for the current situation. These actions are then acted upon, and may include giving orders to subordinate command agents.

Initial results have shown that by considering only a few actions the agents are capable of generating complex plans.

2. Introduction

The cost of performing live military exercises to provide training for future commanders is becoming increasingly prohibitive. The complexity of training higher ranking officers in strategic thinking in battle

could require an exercise involving thousands of troops, which except occasionally is infeasible.

Hence, synthetic environments, (or a virtual battlefield) are increasingly being used to simulate battlefield entities and their interactions to provide commanders with their training experience. Here the trainee commanders control their forces as in a real exercise, by issuing orders, but the behaviour of the units and/or entities are computer generated. The opposing forces are similarly controlled by an operator controlling a number of Computer Generated Force (CGF) units.

These simulators inter-operate (for example using the Distributed Interactive Simulation, DIS, protocols) to give the appearance that the commander is competing against a real adversary. Although this training method may be more cost effective than live exercises, it still requires many more operators than trainees. Additionally the intelligence level of entities in many systems is such that the operator frequently has to intervene to correct inappropriate actions of the units.

It is therefore desirable to increase the level of automation and intelligence of the CGF systems to reduce the number of operators and the amount of operator intervention required. This has led to a large number of CGF research activities (see Ceranowicz 1994; Courtemanche and Ceranowicz 1995; McEnany & Marshall 1994).

Clearly a CGF system should represent realistic behaviour with minimal operator intervention. In the complexity of the battlefield the use of reactive systems or drill procedures is not ideal because it does not:

1. account for enemy positions or intentions,
2. utilise the environment to its best effect (e.g. considering the GROUND).

Hence, a more general planning system which can consider the outcome of its actions is seen as a more appropriate solution to aim at. To achieve this aim, we are investigating the use of "broad agents" (Bates, Loyall, & Reilly, 1991) for implementing intelligent battlefield entities.

This approach aims to remove the need for detailed doctrinal Combat Instruction Sets (CISs) by raising the level of decision making. Instead the agent would have knowledge of the principles of manoeuvres and constraints about the world allowing orders to be issued in the form of objectives to be attained thus endowing the agent with a measure of initiative. This methodology is in keeping with the current British Army doctrine of mission command.

This paper examines some important issues in developing an effective CGF system. An outline is given of the approach adopted at DRA Malvern to provide plausible agent behaviour, and some results are presented.

3. Characterising the Problem

The physical properties of entities (such as movement, fire rates, ballistics, etc.) can be modelled correctly using appropriate mathematical models. However, modelling human behaviour is much more difficult because representative models are difficult to specify and are computationally too expensive to operate within the usual time constraints of a CGF system. However in mitigation more than one course of actions may be acceptable in a given situation, and it is generally sufficient that the CGF chooses a plausible course of action. Hence, a heuristic methodology of planning is suited to modelling behaviour (see Meliza and Varden 1995).

The process of planning within the battlefield is a complex process. The agent has to take account of many factors, including:

- Movement of enemy forces.
- The Intention of the enemy force.
- Uncertain, incomplete and out of date information ("Fog of War").
- Terrain features.

Conventional game theory is inadequate because the number of possible actions is unbounded.

We have identified five principles which a CGF should exhibit if it is to realistically represent the real world:

3.1 Appropriate Actions

The agent should be capable of acting in any situation it may encounter. A CGF system based on CIS or drill procedures could easily find situations for which it does not have a procedure for, in which case it may not perform any action. Worse is a combination of situations (such as encountering a minefield while under air attack) where a procedure must exist for the combined situation otherwise an inappropriate action could be executed (such as scatter).

Our approach overcomes these problems by considering the environment and the effects different courses of actions would produce and so chooses the most appropriate plan.

3.2 Act at Tempo

One of the most important considerations in military operations is that of Tempo. Tempo is where a force executes sequences of events, or performs several activities at once, more quickly than the opposition is able to keep pace with, and hence becomes overwhelmed. Therefore, to prevent being outpaced (or out manoeuvred), the commander must be capable of quick responses to situations whilst maintaining an overall scheme, so maintaining purpose.

Our implementation has the facility for the agents to respond to a situation quickly by invoking a reactive behaviour, but still developing (in the background) an overall plan of action.

3.3 Co-operative/ Co-ordinated Behaviour

In any operation or activity it is essential that the agents act co-operatively (or mutually support one another) for greater effectiveness of the unit. This holds whether on an offensive manoeuvre (e.g. covering fire) or a movement operation, which generally requires good co-ordination of the agents. For example, only one entity can cross a bridge at a time.

Effective co-operation results from a hierarchical command and control structure which includes the re-allocation of roles following attrition.

For this reason and because it corresponds to real C² systems, we have chosen to use a hierarchy of command agents.

3.4 Operator Interaction

Ultimately the CGF system must interact with a human operator who provides the overall battle plan, and can intervene to modify the actions of a unit if required (e.g. to create a situation to test the trainee). Therefore, the method of communicating information between the operator and the entities must be in a form which is clearly understandable, and in a familiar format (e.g. see Ceranowicz, Coffin, Smith, Gonzalez and Ladd).

The tool-kit developed enables any communications method to occur between the agents and the outside world. This is achieved by a "plug in module" which translates messages between the outside world format and that required by the particular agent message handling implementation.

3.5 Plausible but not Predictable

In order for the trainee to be convinced they are battling against a real opponent, the behaviour of the entities must be plausible, both in the short term (short term behaviour) but also in the long term (overall strategy). The more cunning the opponent is, the more intelligence and initiative is required of the trainee to triumph.

However, the CGF entities must not be predictable, and always do the same action between similar situations. If this were the case, the trainee could learn what action to expect from their opponent, which is obviously undesirable.

Our approach intends to overcome these problems, by considering different courses of actions and selecting either the best, or one from the 'n' best plans and thereby introducing uncertainty.

4. Tool-Kit Implementation

The framework for the agents has been developed in collaboration with Birmingham University as a tool-kit "SimAgent", written in Poplog (Sloman and Poli 1995). The SimAgent tool-kit allows multiple agents to run, controls the message passing between them and allows either simple internal simulation, or control of a remote simulation.

4.1 Requirement

In order to support the complex internal information processing and the casual interactions of agents (or objects) both external and internal behaviour is required. In general a general idea of the desired architecture is known before hand, however it was unclear at the start of the project which particular architecture would support the functionality of a broad agent.

The tool-kit therefore needed the facility to support different architectures between the agents, but also the agent itself may require a number of sub-architectures to support all its functionality. Thus the ontology space imposed by the tool-kit should not be restricted.

Also, the agents need to interact with each other, but also (especially in a DIS environment) interact with other agents (or entities) in the world. Therefore, the agents must perform some simulation (physical). This can be achieved using internal modules to the tool-kit, or by enabling the agents actions to control a separate simulation system.

The tool-kit is based on the POPRULEBASE library provided within the POPLOG development environment, and which also provides rapid prototyping capability.

4.2 Scheduling

The tool-kit scheduler is responsible for the correct running of each agent (or object). The top-level scheduler procedure is presented with the list of agents, and the maximum number of time slices for each agent.

The scheduler runs in a two pass operation. Firstly, it allows the agents to perform any sensor operations, read messages from other agents, and perform internal processing (behavioural modelling). Secondly, it passes any messages between the agents, and runs any external actions, such as move, turn, etc.

This ensures the behaviour is generally independent of the order of the agents, because all agents get to perform sensor and thinking activities before the actual state of any entity is changed.

4.3 Agent Mechanisms

Each agent has an associated collection of rule sets. Each rule set is a collection of condition-action rules, that interact via one or more databases. Hence, one rule set might be concerned with sensory perceptions, another rule set may be involved in planning activities, and so on. The rules can switch between databases, push them on a stack, restore them, etc. (c.f. with SOAR, see Laird, Clave, Erik and Roberts 1993). Rules also can invoke other rule sets.

Although learning is not included in our implementation, it is supported in the tool kit. A rule can introduce new rule sets, or create new rules within a rule set. The rules are not limited to a particular style, they could invoke a theorem prover, such as Prolog, also since the rules can invoke Pop-11, it is (with Poplog Pop11) possible to invoke other languages such as C or C++.

The tool-kit also has the ability of simulating constrained thinking time (or resource limited planning) by constraining the amount of processing any one agent can perform on a cycle.

4.4 Actions

Inevitably the agents will want to perform some actions based on their motivations, thus the agents ultimately need to perform physical actions. The tool-kit scheduler allows the objects to execute actions during its second pass either by updating entries within its database, or its object slots.

However, the tool-kit has been extended to enable the agents to control the physical operation of entities running on a remote simulation using simple command and control messages. These messages also allow information from the entities sensors to pass back to the agents. Thus the mental and physical modelling of the entities is entirely separated.

4.5 Conclusions

The tool-kit provided enables a general approach to be adopted for the implementation of the agents. It does not present undue restrictions on the architecture employed to achieve the desired behavioural representation. By providing the facility to execute a number of agents together enables the modelling and execution of small military groups, such as troop (or platoon) and squadron (or company) on a single workstation.

The addition of the facility to enable the agent to control remote entities has obviated the need to redevelop the physical simulation part of the entities (e.g. motion dynamics, sensors, etc.) by using our existing simulation software. This also enables the entities to interact with other simulated entities by using the DIS protocol.

Figure 1 shows the relationship between the agent objects, the rule sets, the tool-kit and remote simulation.

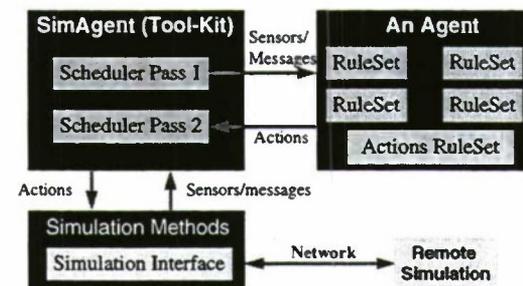


Figure 1. Tool-Kit Overview.

The first pass of the scheduler allows all the agents to process sensory information and incoming messages, and run their behavioural methods. This could generate a number of actions in a number of agents. Only when the scheduler does its second pass are the actions executed, using a (user defined) API to the simulation methods.

If required a different remote simulation could be used by simply replacing the Simulation Methods Module. Since the actions and sensors use a (user) defined structure, provided the new module communicates with the tool-kit using the same format, no changes are required to the agent's rules. Similarly the physical simulation could be entirely contained within the simulation methods module.

5. Agent Implementation

As mentioned the entity intelligence is based on a "broad agent" implementation within a military Command and Control (C²) hierarchy modelling the structure of military units. A broad agent (Bates, Loyall, & Reilly, 1991) is designed to have a broad but shallow range of capabilities, instead of a limited but very detailed set of behaviours. Our intention is to implement a planning framework based on the military doctrine of Mission Command and avoid detailed CIS for every situation. This enables the

agents to create plans of action taking into consideration their objective and their estimation of the ground. The C² hierarchy enables commands to be devolved from high level commanders to subordinates and so each agent only considers the appropriate level of detail.

The scenario being developed to demonstrate the agents' behaviours is based on a battle group formation, shown in Figure 2.

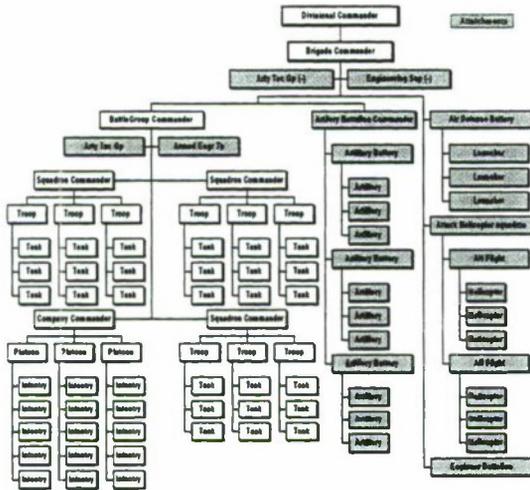


Figure 2. Example of a BattleGroup Hierarchy.

A broad scenario was adopted because it encourages a broader approach to implementing the intelligence because the units will have to think about different types of friendly or enemy units on the battlefield, and their associated roles. The aim is to implement simpler intelligence initially which can think more generically about the battlefield, rather than highly intelligent units which can only reason about a specific type of unit and limited number of situations. Additionally using an existing C² hierarchy directly relates units at each point to the real world, making it easier to extract the behaviours from subject matter experts (SMEs). This also helps with validation of the behaviours, because a military officer can watch the actions of units and by using their experience evaluate the realism. Because the units are simulated separately, the force can be distributed across a number of computers and a larger number of entities can be simulated. Physical interaction can be achieved using DIS protocols for instance, and a separate Command and Control language (which could be the Command and Control Simulation Interface Language, CCSIL) used for instructions and messages.

We have not embodied learning behaviour into the agent in the present implementation because it was felt that a lot of behavioural principals could be extracted from Army Field Manuals, and Doctrine Manuals.

The current implementation extends to one tank squadron of the Battlegroup of Figure 2., shown in Figure 3.

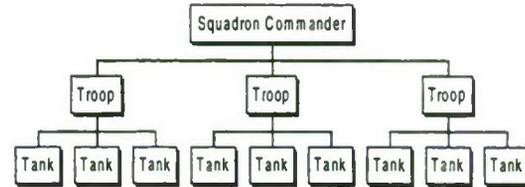


Figure 3. Squadron Composition.

Each agent within the hierarchy is based on a broad agent, the basic design is shown in Figure 4:

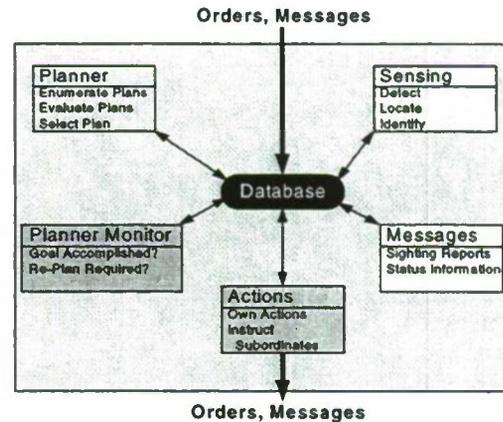


Figure 4. Broad Agent Design Concept.

The fundamental properties of this design are:

1. It contains a central database, through which all the modules communicate. This database can be partitioned into a number of sub-databases each holding a set of related data, allowing searching within the database to be performed much more quickly, and potentially to distribute the database across processes (locally or over a network).
2. Individual modules can be identified to perform fundamental tasks. Although these can be separated their operation may be related to another module, for example, plan monitor which

monitors the progress of a plan relies on sensory information. However, separating functionality enables parallelism of the modules.

3. The modules only simulate the agent's intelligence and do not perform any actual physical modelling. If the agent wishes to perform an action, for example change the heading, it sends an instruction to the physical simulator with the request. The intelligence gets confirmation back about the action via its sensors. The advantage of this approach is the intelligence modelling can be separated from the physical simulation, and so allows the re-use of existing physical simulators. The control is replaced with our intelligence. It also allows the two parts to be run on separate processors (locally or over a network).
4. The design of the intelligence is then generic to any position in the C² hierarchy.

It should be noted that a single battlefield entity (such as a tank) is modelled as a single agent (i.e. the commander, gunner and driver are aggregated). The overall operation of the unit seen externally appears as a single intelligence, hence the need for separate modelling of the individuals within the tank is not necessary. Taking the next command level, the troop commander, also resides in a tank. This tank performs the same functions as a subordinate tank except it is also able to command other tanks, and so has extra functionality. This is emulated within the broad agent design by additional modules which performs troop command thinking. This can be visualised in Figure 5.

If during a battle the troop commander is destroyed, the second in command of the unit takes charge of the unit. This is achieved by the new troop commander gaining the troop commander capabilities (i.e. by adopting the extra rule sets) and continues the campaign. Note, the new troop commander does not obtain the database from the previous commander, but, in accordance with military doctrine already knows the objective in hand.

Initially all agent functions ran internally to the Tool-Kit, which obviated the need for real time constraints. However, when the link to the external physical simulation was established, it became important that the agents ran in real time. This is achieved by constraining the amount of processing an agent can perform on a cycle. However, the consequence is that

plans may take longer to generate. In situations where

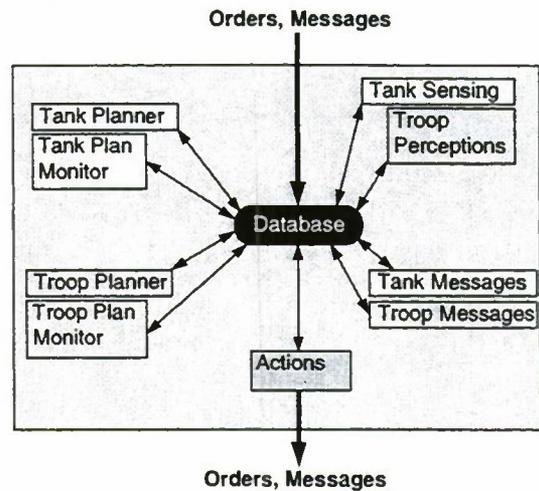


Figure 5. Example Troop Commander Architecture.

immediate action is required this would not be acceptable, hence, the agent would need to fire either a reactive behaviour, or a very simple planning action to deal with the immediate problem.

The behaviour of the tank agents is observed by using them to control tanks inside a simple simulation linked with the agent Tool-Kit in a small tank battle scenario.

Currently only the tank agents are able to perform planning (Baxter and Hepplewhite 1996); the troop and squadron commanders currently follow prescriptive decomposition of commands. For instance, the Squadron Commander can accept an order to occupy an area of terrain, which is devolved to orders for each troop commander to occupy a different part of that area. The troop commander determines the heading and speed for the troop and informs each tank individually.

6. Agent Planning

Individual agents are responsible for choosing targets for their weapons and controlling their own motion within the framework set out by the troop commander. To do this they take orders from the troop command agent as a set of guidelines and constraints. Plans have to be constructed which fulfil the goals set by the troop commander as far as possible without violating any constraints. An example of a constraint could be a boundary line that the agent must not cross, perhaps because this area has been assigned to a different troop. The desired

speed and heading of the troop as a whole is taken by the tank agent as a guideline to derive goals in space and time for the tank agent.

6.1 Planning Framework

The guidelines and constraints supplied by the troop commander are used to derive a short term positional goal, a planning horizon (the maximum time to which plans should extend), and a target time by which the positional goal should be reached. Without the presence of obstacles or opposing agents these settings cause tank commanders to move together as a group in a formation dictated by the troop commander.

The method which has been adopted uses the present state of the battlefield to suggest a number of discrete actions for a tank agent to consider. Each of these actions has an associated cost in distance, time and risk which are combined together to give an overall cost by use of a weighting function. Actions may be rejected because they violate one or more of the constraints set down by the troop commander. Combinations of actions with their cumulative costs are made into plans and an optimal plan selected by an A* search (see Russell and Norvig 1995). This enables the search space to be restricted to actions which, in general, seem applicable to the agent's present situation. By basing the cost on a detailed consideration of the effects of the action, situations in which the action is inappropriate can be identified. The generation of multiple actions also provides for the possibility that a slightly sub-optimal choice could be made to prevent the agent's actions being too predictable.

6.2 Generating Actions.

Five actions are currently available to a tank commander agent. They can be described as advance, assault, retreat, run away, and face. The advance action involves moving toward the goal with a speed and heading to ensure the agent is within its assigned formation position at the end of the motion.

The other actions are more complicated. They are based upon the fact that a tank's frontal armour is much stronger than its side or rear armour so the best approach to an opposing tank is to face it, thereby reducing the probability that a hit will be destructive. The actions therefore try to combine facing the enemy with progress towards the goal.

To help in generating actions from this knowledge the agent first makes a "threat map" which identifies threat with respect to heading based on the proximity of hostile tanks. Using this threat map two potential actions can be directly generated; facing the maximum threat and remaining stationary, the "face" action, or reversing, the "run away" action. The assault and retreat actions try to find a "safe" direction which also lead towards the goal. This is done by multiplying the ratings of the threat map by a value proportional to the difference between the heading and the heading direct to the goal (for advance) or directly away (for retreat). Selecting the headings with the maximum values after this process (so long as there is a non-zero value present) gives two suggested actions which should give motion towards the goal but on a heading so that the risks to the tank agent can be minimised. Presently each action is considered to last for ten seconds, after which new actions will be considered.

6.3 Costing Actions

For use in the A* search the cost of selecting an action as part of the plan must be evaluated along with an estimate of the cost to reach the goal after the action has been completed. The distance and time costs are simple to derive but an accurate measurement of the risk to which an entity is exposed is much harder to evaluate. At its most extreme this would require a statistically significant number of simulations of the execution of the plan to be carried out and the proportion of instances in which the agent was destroyed to be found.

Instead an estimate of the risk is found by making several assumptions about the course of events. The vehicle is assumed to make the move to the end point of the action in a straight line and assumes all other agents continue in straight lines. The risk involved is calculated by assuming all hostile agents within range may fire upon the agent with a probability based upon the relative strengths of the two sides involved in an engagement. The risk is therefore a combination of the probabilities that each enemy agent will choose (or be able) to fire on the planning agent, that the shot or shots will hit and that a hit will penetrate the armour and destroy the tank. This gives an overall probability of destruction during an action.

The estimate of the risk in reaching the goal after the action is taken to be zero, thus ensuring it is an underestimate and guaranteeing the optimal set of actions will be selected by A* search.

7. Results

Initial tests have been carried out using two opposing troops of three tanks whose starting positions and final goals are such that they move into view of each other with intersecting paths if no avoiding action is taken. By altering the relative weights attached to time taken, distance and exposure to risk, different behaviour patterns occur. Thus a "personality" can be attributed to a commander in terms of their cautiousness or risk taking behaviour.

In the risky vs. risky scenario shown in Figure 6 the two groups come into sight of each other when they reach A. When they come within range at B both groups launch into an immediate assault and a fight ensues. Neither group is prepared to postpone its actions in order to avoid a fight but do alter their approach to the goal to face the enemy. By the time the groups have closed to C all the red tanks and one blue tank have been destroyed. The remaining blue tanks continue to their goal.

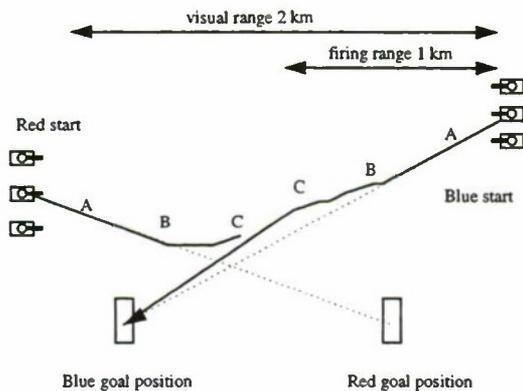


Figure 6 Combat between two groups of 'risky' tanks.

In the risky on cautious scenarios after a brief initial engagement the cautious group retreats out of range and stays clear of the more aggressive troop, even though this delays them considerably.

Two cautious troops, shown in Figure 7, tend to hover around the limits of the maximum firing range trying to work their way around each other and occasionally exchange shots. Neither group is willing to expose itself to the opposite side's fire in order to achieve the goal. When the groups come into firing range at B both retreat but the red force has to take a large detour since the tanks cannot move directly to their goal without exposing their flanks to fire from the blue troop. The blue troop on the other hand

simply delays its advance and moves towards the goal when the red troop has retreated out of range.

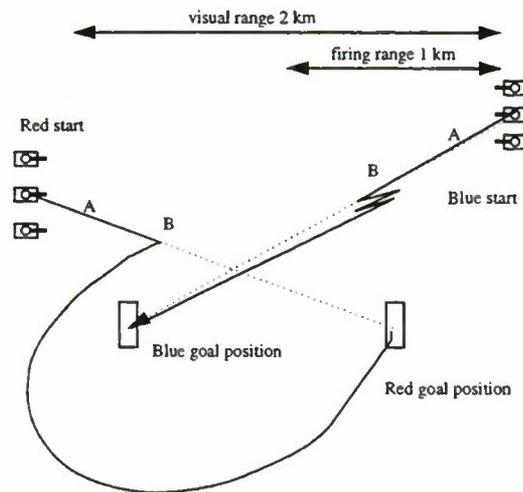


Figure 7 Combat between two 'cautious' groups.

These scenarios demonstrate that appropriately different behaviours can be achieved by simply changing the risk parameter the agents find acceptable. This shows that different command styles (ranging between cautious and the reckless) can be easily emulated.

The behaviour of the agents have been viewed by a military advisor and agent responses to enemy tanks accepted as plausible. The main deficiency was recognised as the lack of terrain utilisation by the agents.

The planning speed of the agents varies considerably depending on the level of activity. When a group is isolated from any opponents, planning is relatively simple and fast. However, when an opposing unit comes into range, the planning process becomes more complex as the agent searches through sequences of alternative moves, to optimise its cost over the next few seconds of the scenario. Consequently the planning becomes much slower.

8. Conclusions

The results we have obtained show that our approach has promise. The generation of actions, based on the current situation allow the potentially infinite search space to be restricted to considering only a few, promising, actions. The costing of these actions by

the use of a mini "simulation within a simulation" enables agents to identify when an action is inappropriate due to the specific conditions under which it would be executed.

We have also seen that the time for agents to plan can vary dramatically as the situation of the battlefield changes. For training and inter-operation with other systems it is essential to operate at a consistent rate, usually at real time. To ensure this real time AI techniques must be used.

9. Further Work

Immediate enhancements are aimed at incorporating more intelligence into the troop and squadron commanders to develop longer term tactical plans. These are intended to be based upon the same framework as the planning for individual entities. It will include simulation of battlefield activity at a more abstract level including simulation of enemy responses. The actions available to individual tanks will be increased to include terrain based activities, such as hiding and seeking cover.

10. Acknowledgement

We would like to acknowledge the support of D/ISIS, UK Ministry of Defence.

11. References

- Bates J, Loyall A B, & Reilly W S, "Broad Agents", 1991, Sigart Bulletin, pgs 38-40.
- Baxter J W, Hepplewhite R T, "Broad Agents for Intelligent Battlefield Simulation", Proceedings of the 6th AI, Simulation and Planning in High Autonomy Systems, La Jolla California March 1996: University of Arizona, pp 199-205.
- Ceranowicz A, "ModSAF Capabilities", Proceedings of the 4th Conference on Computer Generated Forces and Behavioural Representation, Orlando Florida May 4-6 1994: Institute for Simulation and Training, pp 3-8.
- Ceranowicz A, Coffin D, Smith J, Gonzalez R and Ladd C, "Operator Control of Behaviour in ModSAF", Proceedings of the 4th Conference on Computer Generated Forces and Behavioural Representation, Orlando Florida May 4-6 1994: Institute for Simulation and Training, pp 9-16.
- Courtemanche A J and Ceranowicz A: "ModSAF Development Status", Proceedings of the 5th Conference on Computer Generated Forces and

Behavioural Representation, Orlando Florida May 9-11 1995: Institute for Simulation and Training, pp 3-13.

Laird J E, Clave B L, Erik A and Roberts D, "Soar User's Manual (V6)", 1993, University of Michigan, Carnegie Mellon University.

McEnany B R & Marshall H, "CCTT SAF Functional Analysis", Proceedings of the 4th Conference on Computer Generated Forces and Behavioural Representation, Orlando Florida May 4-6 1994: Institute for Simulation and Training, pp 195-207.

Meliza L L and Varden E A: "Measuring Entity and Group Behaviours of Semi-Automated Forces", Proceedings of the 5th Conference on Computer Generated Forces and Behavioural Representation, Orlando Florida May 9-11 1995: Institute for Simulation and Training, pp 181-192

Russell S J and Norvig P, "Artificial Intelligence: A Modern Approach", 1995, Prentice-Hall, Englewood Cliffs New Jersey.

Sloman A and Poli R: "SIM_AGENT: A toolkit for exploring agent designs", August 1995, ATAL-95 Workshop on Agent Theories, Architectures, and Languages, IJCAI-95 Montreal.

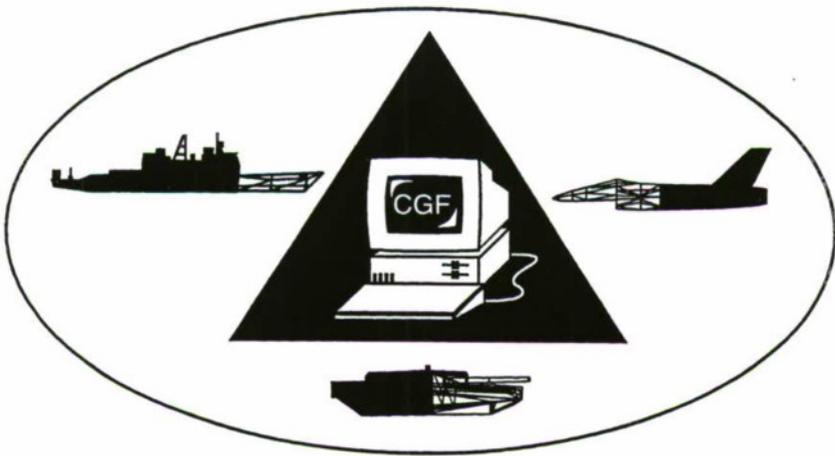
12. Authors' Biography

Richard Hepplewhite read Engineering and Computer Science at Oxford University. After graduating he took employment at the U.K. Defence Research Agency on Autonomous Command Agents. His research interests include command and control hierarchy, and terrain reasoning methodologies.

Jeremy Baxter gained an Engineering Science degree from Durham University and then completed a PhD in Robotics, also at Durham. He has worked at DRA Malvern since 1994 on AI for CGF. His research interests include Fuzzy Logic, planning and search algorithms and real time plan execution and monitoring.

© British Crown Copyright 1996/DERA

Reproduced with the permission of the controller of Her Britannic Majesty's Stationary Office.



Mission Planning and Coordinated Execution for Unmanned Vehicles

Patrick G. Kenny, Edmund H. Durfee and Karl C. Kluge
Artificial Intelligence Laboratory
The University of Michigan
1101 Beal Ave
Ann Arbor, MI 48109-2110
{pkenny,durfee,kckluge}@umich.edu

1. Abstract

The battlefield of the future will be comprised of both human- and computer-controlled entities, where the hope is that unmanned systems will increasingly take on the higher-risk missions. Realizing this vision involves the construction of entities such as unmanned ground vehicles (UGVs), and the development of mechanisms by which human commanders can convey the intent and parameters of a mission to UGVs and can monitor/correct the performance of UGVs. At the same time, the commander cannot be expected to hand-hold the UGVs, so the UGVs need the capability to replan and improvise (in a doctrinally-appropriate way) so as to fulfill the mission goals in a coordinated manner, with minimal human intervention.

Our research has been focusing on the development of tools for planning, execution, and coordination of multiple UGVs at a strategic mission level. The interface to a human commander permits the specification of a mission at an appropriately strategic level. Our underlying tools work with the commander to elaborate the mission sufficiently to begin its execution, and then during mission execution our mechanisms allow the flexible achievement of various objectives based on the details of circumstances encountered.

We have developed a system that is comprised of multiple instances of a procedural reasoning system, an interface for the specification of high level military missions, tools to help plan routes, formations, and observations, and to assimilate various kinds of information and monitor plan execution.

2. Introduction

The ability for users to generate multi-vehicle plans which perform realistic behavior and coordinated control of unmanned vehicles without specifying the

low-level detail that is required to control the robotic vehicle is and will be of great importance in the usefulness to deploy and use robots. The ability for a robot to process and carry out the goals in the plan in a reactive manner will also be crucial for the robot to accomplish its mission.

This paper describes a system that was developed for the Unmanned Ground Vehicle (UGV) project to initially add mission planning and reactive re-planning for the UGV vehicles, then later as a alternate design approach for the control and coordination of robotic vehicles in a simulated environment. The system is composed of a high-level mission planner which aids the user in composing a mission plan by placing militarily significant information on a Graphical User Interface. The objectives so specified are then decomposed into a set of plans for multiple vehicles. The second part of the system is a set of vehicle processes which execute and monitor the generated plans in a simulated environment, ModSAF.

The underlying architecture that is used to control each of the components is a procedural reasoning system called UMPRS (Lee et al, 1994). UMPRS allows for the system to generate realistic behavior by encoding military doctrinal knowledge inside the reasoning system which is called upon when the appropriate context arises.

This paper starts out by describing the overall system architecture and all the components of the system. It then describes the mission planner functionality, where the user enters the mission and the planner decomposes it into a set of vehicle plans. Following that the paper then describes the agent architecture processes and related tools. The last section describes the execution and reactive re-planning that can be performed by the agents in the system. We then conclude with an analysis of the architecture and how

and where it can be applied in the computer generated forces arena.

3. System Architecture

The system we have designed and built is composed of many modules interconnected and communicating amongst each other. At the top of the system is a mission planner connected to a GUI, the middle of the system is composed of the vehicle process and tools and the bottom is the vehicle simulator, ModSAF. See Figure 1 for a view of the system architecture.

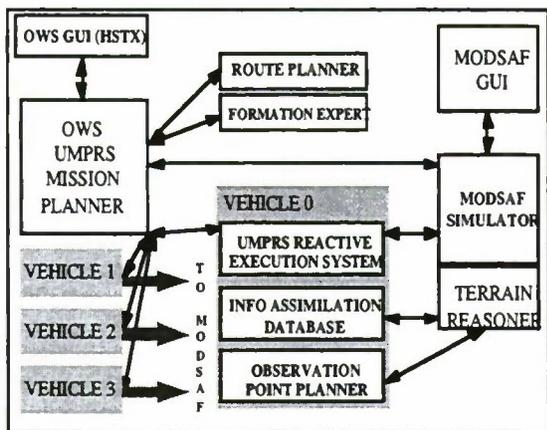


Figure 1. System Architecture

Following is a brief description of each of the modules in the system.

3.1. UMPRS

In order to perform realistically in a real or simulated environment a vehicle should execute tasks in a real time and reactive manner. We have developed a Real Time Planning and Control Procedural Reasoning System, UMPRS (Lee et al, 1994). UMPRS is based on SRI Procedural Reasoning System (PRS) developed by Georgeff et al, (Georgeff & Lansky, 1986). UMPRS is a general purpose reasoning system, integrating traditional goal-directed reasoning and reactive behavior.

UMPRS is well suited for use in robotic applications because it allows the robots to pursue long term goals by adopting pre-determined procedures based on context and not blindly following a prearranged plan. It is also well suited for use by robots in a reactive environment where it can switch out of its currently executing procedure and invoke another appropriate procedure when the situational context changes. For example if a vehicle is traveling down a road executing a move-to procedure that has context that

states that there are no enemies in sight, and then it spots an un-friendly vehicle, the context for the current world state will be changed the current procedure will be switched out and another procedure will be invoked to deal with this new change in context.

UMPRS is composed of five components, see Figure 2:

Database -

A database called a world model that contains the beliefs and facts about the world. Initial facts are asserted at the beginning of a UMPRS program, other facts and beliefs can be either asserted or retracted by KA's, which is explained below.

Knowledge Area's -

A set of declarative procedure specifications that describe how to achieve a system goal or query. A KA consists of a purpose (a goal, test or query or action) for executing the KA. The context in which the KA is applicable and a body which is viewed as a directed graph in which nodes represent states in the world and arcs represent actions or subgoals. The body may consist of goals to achieve or maintain, goals to test, subgoals, branches, assertion or retraction of beliefs or primitive function to be executed, this is just a partial set of action, for full set refer to the UMPRS User Documentation (Huber et al). A primitive function can be a function to control a robot, send communications or perform some calculation. primitive functions are the low-level interface to the real world and are generated by the user.

Goals -

UMPRS maintains a set of current goals to be realized, KAs may place additional goals or subgoals into the system by attempting to achieve some action. A top level goal is different from a subgoal in that the system will continue to pursue the top level goal until it is satisfied, but goals within KA bodies are not persistent.

Intention Structure -

The intention structure acts as the run-time stack of the system. It keeps track of the progress of each high-level goal and all of the subgoals. The intention structure suspends, resumes, cancels and proceeds with the execution of goals in much the same way as an operating system. The intention structure maintains information about what KAs are currently

active, as well as what actions in each KA are to be executed next.

Interpreter -

The interpreter is what controls the execution of the entire system. Whenever there is new or changed information in the world model or goal list, the interpreter determines a new set of applicable KA's, called a SOAK set, to pick the next appropriate KA to be placed in the intention structure. When there is no new SOAK being generated, the interpreter checks the intention structure for the currently active KAs and executes the next action. If this action changes the goal list, by creating a subgoal or satisfying a goal, a new SOAK is created and the cycle starts over. If a new SOAK is not created then the next arc in a KA is executed.

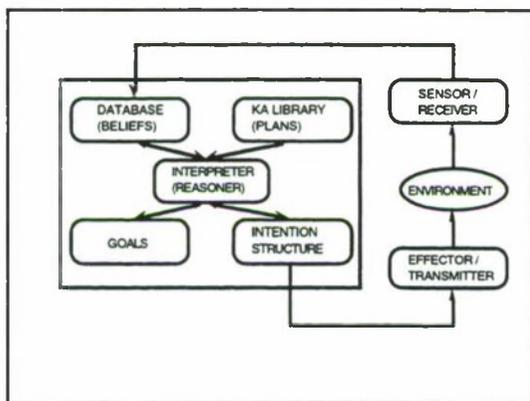


Figure 2. UMPRS Architecture

In the system that we have developed, UMPRS is used both as a mission planner and as a vehicle plan execution, re-planning and monitoring process, both of which are described in later sections of the paper. For a UGV application, the plans, beliefs, contexts, and goals are based on standard operating procedures as defined in military doctrine.

3.2. Mission Planner

At the top of the system is a mission planner composed of UMPRS and a KA library that describes how to decompose the mission into a set of goals for the vehicles to achieve. The mission planner is connected to a Graphical User Interface where the user enters military significant icons and mission data. The mission planner has tools available for the decomposition of the plans; these include a route planner, a formation expert and an observation points

planner. The mission planning process is described in detail in a later section.

3.3. Operator Work Station (OWS)

The OWS GUI was designed and built specifically for the UGV project. The OWS consists of a map interface able to read in various formats of DMA terrain data. Tools to display and analyze map data, get distances, perform line of sight calculations, plan routes, add military measure icons and perform other various UGV specific planning and execution monitoring. For our purposes, we are using the OWS as a graphical interface to the mission planning processes.

3.4. Vehicle Processes

The vehicle processes in the system execute and monitor the plans and goals generated from the Mission Planning process. There is a separate vehicle process for each vehicle used in the mission plan. Each vehicle process consists of a UMPRS process and KA library that describes how to accomplish goals, an information assimilation database and an observation points planning process. The vehicle process and execution are described in more detail in a later section of the paper.

3.5. ModSAF Vehicle Simulator

The ModSAF vehicle and forces level simulator is used as a testbed for the execution of the vehicle processes. Although in the UGV project we have access to multiple Hummer robots, the cost and time it takes to run missions on the vehicles is prohibitively large compared to that of running scenarios on a simulator. Since the actual UGV vehicles are limited in their sensing and what they can do, we are able to run more complex behaviors and missions with the simulator. We can set the simulator and system up to be run as a friendly or enemy force and have them each perform exercises against each other.

3.6. TCX Communications Package

The last part of the system is the communications package that ties all of the processes together. We used a communications package, developed at CMU called TCX [Fedor]. Through TCX we are able to send reliably data structures over the internet.

4. Mission Planning

The process of describing the low level details of robotic plans can be an enormous task, especially if there are multiple vehicles used in a mission. Military planners like to think in high level terms when planning missions, not low-level robotic details. The gap that exists between the military planners and the detailed plans needed to be executed on a robotic vehicle are filled in by the mission planning capabilities of the system.

The mission planning part of the system is composed of a UMPRS process and a set of KA libraries, an OWS Graphical Interface process, a route planner tool process, a formation tool process, and an observation points planner tool.

The user starts a planning cycle by entering on the graphical interface the military specification of the mission and the associated military measure icons. For example to plan a reconnaissance mission for two vehicles the user would enter an assembly area measure for the vehicles to start, a few checkpoints that the vehicles need to pass through and an observation post where the vehicles would perform a recon. See Figure 3 for a planning example.

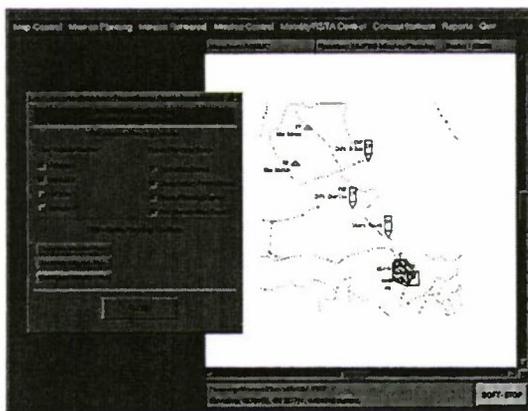


Figure 3. Mission Planning Example

The icons and associated data would then be sent off to the mission planning process to be elaborated. The icons could have been entered in any arbitrary order, and the sequence and strategies for pursuing the measures might not be obvious purely from the map. As a result, in order to resolve ambiguities the mission planner draws on its library of templates or KAs for operations, or can call on the user to provide such a template. By matching military symbolgy to the template, the mission planner formulates the

sequence of action and goals that must be carried out by the robotic vehicles.

As it elaborates the mission, the planner can call on more specialized tools to formulate portions of the robotic plan. These include tools for planning routes, planning appropriate reconnaissance points along a route, and planning formations for vehicles to travel in when necessary.

The detailed steps of how to elaborate the data from the mission specification are defined in the UMPRS KA library that the mission planner uses. For example these can be simple steps such as move from one checkpoint to another, to complex steps such as planning a set of bounding overwatch points, where one vehicle watches as another moves along a route, then they reverse roles. As more KA sets are added to the library, the system will be able to plan more and complex behaviors and missions. The degree of autonomy does not necessarily have to be at the vehicle level. Since UMPRS KA's are a set of procedures for accomplishing a certain goal, the goal may be a high level goal such as plan for an assault on a position with many groups of vehicles, where the reconnaissance task is just a small part of the overall goal.

The elaboration of the plan does not stop at the mission planning level; indeed, once the plan is sent to the vehicle process it may elaborate the plan farther to develop more detailed control. Our architecture makes no commitment to which degree of autonomy is instantiated at any given time; rather it will depend on the particular knowledge and procedures provided to the various agents.

The military missions that we have developed are centered around reconnaissance tasks executed by a team of two or four vehicles. Some of the subtasks may include bounding overwatch, deep range recon, cooperative vehicle sensor observations and reacting to unknown or enemy threats by replanning. Reactive tasks can be replanned by the vehicle, such as when a vehicle needs to find an alternate route to a goal point, or in some cases it may require the assistance of the operator or mission planner, depending on the level of autonomy in the system, to issue a new plan when the team cannot achieve a goal or mission. The missions are based closely on military doctrine for executing reconnaissance operations. This information is explicitly inserted into the mission planning system or vehicle process as procedures and goals, in the form of KAs to be achieved. As added

knowledge is stored in the system, the vehicles will be capable of performing more detailed and complex missions.

5. Agent Architecture

Once the plans are generated for each of the vehicles in the system, they must be run on an actual robot or agent. This section describes the agent architecture that we have developed to control each autonomous vehicle. Like the mission planning process, the heart of the autonomous vehicle agent is a UMPRS process and a KA library. In the current system the mission planner creates the vehicle process, or processes if there are several vehicles, after it has elaborated the plans for each vehicle. When each vehicle process is invoked it opens a connection to any other vehicles also present, opens a connection to the mission planner and a connection to the vehicle simulator. It also runs a vehicle level information database, and an observation points planning tool. The information assimilation database is used as a blackboard where it can make requests for certain types of information and once that information becomes true, the process will send that information to the vehicle. The observation points planning tool is run afterward as a reactive behavior when the vehicle detects an enemy.

UMPRS is well suited as a device to control autonomous agents because of its ability to reactively switch behaviors based on environmental context. In the system we have developed, vehicles are given a plan of goals that need to be executed, and a set of reactive behaviors when they detect changes in the environment. The reactive behaviors are called survival behaviors and have a higher priority than any of the goals in the plan. Currently in the system, the only context that will invoke the reactive behavior is when a vehicle detects an enemy vehicle. In that case the vehicle retreats to its last safe position it knows about. If there are a group of friendly vehicles and one detects an enemy, then that one communicates to the other vehicles that it has detected an enemy and they should retreat, while it plans for further observations with the observation points planning tool.

5.1 Information Assimilation Database

Groups of cooperating processes or agents need a method to communicate partial results to each other and a shared model of various aspects of the environment. The blackboard model (Nii, 1986) is a common method used in AI research to achieve these

ends. The blackboard is a database which is connected to all the cooperating processes, allowing them to share information and coordinate their problem solving efforts towards common goals. The CODGER system (Stentz, 1990) was a variant on the blackboard architecture developed for use in the ALV project research at Carnegie Mellon. While traditional blackboards explicitly scheduled which process would be active next based on the contents of the blackboard database, CODGER did not perform active process scheduling. Instead, synchronization primitives available through CODGER permitted process synchronization.

The system described in this paper uses a CODGER-like blackboard to store the shared information about the world used by the different processes in the system. Our goal in future work is to extend the blackboard formalism in two significant ways. First, the types of military missions which UGV-like vehicles are expected to perform may cover large areas of terrain (100 or more square kilometers). As a result, efficient spatial indexing of objects with geometric attributes will be crucial to efficient database access. Second, there is an important distinction between an area of the blackboard's "map" not having a certain type of object in it and the corresponding area in the world not having such an object in it. Representing the distinction between absence of data and absence of enemy vehicles (for example) is crucial for task performance. This requires keeping track of which sensors have examined which patches of terrain (and how recently the examination was performed), which again involves choosing appropriate spatial representations for efficient database operation.

5.2. Observation Points Planner.

Intelligent autonomous or semi-autonomous agents, whether in a synthetic environment or the real world, need to be able to actively plan deliberate observations of their environment. This is an area which has generated work in the areas of computer generated forces (Van Brackle, 1993), robotic semi-autonomous vehicles (Cook et al, to appear; Kluge et al, 1994), and planning tools to assist humans (Musman et al, 1994), with the same concepts potentially applicable across all three domains. Our work has focused on the problem of planning sequences of observations to achieve a specified level of certainty with respect to some hypothesis. The initial domain studied was the problem of planning a sequence of observations of a stationary target in order

to achieve a specified accuracy of localization of the target's position. The path the observing vehicle was to follow was preplanned and given as an input to the observation point planning (OPP) module. The goal of the OPP module is to find a set of observation points which should maximally shrink the target position uncertainty given the number of observations made.

Planning such a sequence of observations can be posed as search in a state space which consists of the hypothesis uncertainty space crossed with the observing vehicle's configuration space. In the 2-D stationary target localization problem, the hypothesis uncertainty space is three dimensional (the orientation of the long axis of the two sigma position uncertainty ellipse, and the lengths of the long and short axes). The observer's configuration space is one dimensional (position along the prespecified vehicle path). By using a simple heuristic (given two sets of observations which achieve the same degree of localization, pick the set which does so after a shorter distance along the observer's path), the state space can be kept to the three dimensions of the target position uncertainty space. The Kalman filter defines the new target position uncertainty state given a current target position uncertainty state, an observation location, and a model of the sensor observation uncertainty. Conventional A* search can be used to generate the set of observations which will maximally shrink the target location uncertainty for the number of observations made.

Plans for future work include incorporating models of target detectability (Shaham, 1988) exploring domains with temporal constraints (stationary target, multiple possible observing vehicles), and exploring domains which involve balancing tactical considerations with the improvement in hypothesis certainty (merging OPP into path planning).

6. Mission Execution

The capabilities of our mechanisms currently outstrip those of real UGVs. Thus, while our work is compatible (and has been partially integrated into) a real UGV system developed as part of the DARPA UGV project, much of our development and experimentation has revolved around simulation. The capabilities that we have implemented on top of the simulator allow us to automatically generate a small collection of forces, good and/or bad, and automatically assign tasks to the vehicles, based on

the high level mission specified by the operator. The simulator that we have chosen to integrate into the system is the ModSAF, modular forces simulator.

After the vehicle plans are generated and sent down to each vehicle process, that process creates an entity on the ModSAF simulator. Currently we use a standard Hummer model for the entity; in the future we would like to be able to make a model of an actual UGV Autonomous Hummer with the same sensor characteristics and movement behaviors. This will help in field testing and running through scenarios with combined UGV and human forces. As each entity is set up in ModSAF, they communicate their status to each other. We currently do not use the communications ability with the simulated radios in the simulator, but instead choose to perform all communications from UMPRS process to UMPRS process. Using the radios in ModSAF would be more realistic, but would also involve more detailed planning to make sure that each vehicle was in line-of-sight or that the radio information was received correctly. Next, the vehicles are assigned tasks according to the goals of the mission: they may move to designated positions, perform RSTA observations, perform RSTA on the fly while moving to a location, or observe the movement of friendly vehicles as they move to a bounding point.

The scenarios that we have run on the simulator consist of simple one vehicle recon missions to four vehicle cooperative recon missions. A typical scenario involving two vehicles performing a cooperative recon as follows. See Figure 4 for an example of a two vehicle recon mission being performed in ModSAF. Each vehicle gathers in an assembly area, and they communicate to each other their status. They wait for the mission planner to elaborate the mission entered by the user. Once that is done, the mission planner sends the plan to each vehicle. Each vehicle processes the plan and finds the first goal point to move to, and the UMPRS process assigns move-to tasks for the vehicles to move to their first points. If they arrive there without detecting an enemy, then UMPRS assigns the next move-to task to the vehicles. During the execution of the move-to tasks if a vehicle encounters the presence of an enemy vehicle, then a reactive behavior will be invoked. In this case, the first vehicle to see the enemy communicates to the other vehicle that it has found an enemy and to go find cover. The vehicle that gets the communication returns to the last safe point along its route. The vehicle that saw the enemy now calls upon its observation points planning tool to

find a better point along its route to observe the enemy vehicle. When the observation points planning tool returns the best set of positions along the route, the vehicle stores away its original vehicle plan and creates a new plan with these new observation points as its new set of goals. The vehicle then executes the new plan, moves to the new positions and performs some observations of where it detected the enemy. See Figure 5 for a picture of the mission after the vehicles have detected the enemy and planned new observations.

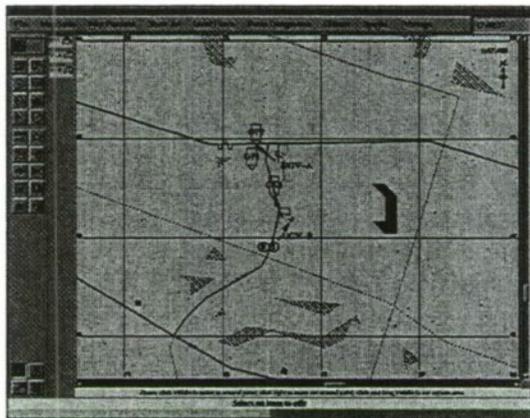


Figure 4. Two Vehicle Recon Mission

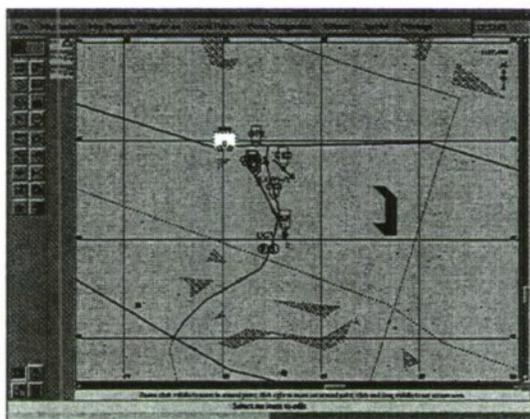


Figure 5. Replanning New Observations

7. Conclusion

An agent-based approach to bridging the gap between operators and robotic vehicles provides significant advantages by embedding knowledge and initiative within a semi-automated mission planning and execution system. In conclusion, the system developed at the University of Michigan explores the possibilities for controlling and coordinating the behaviors of multiple UGVs. The ultimate goal is to put such vehicles at the disposal of military

commanders who can task these systems in very much the same way as they might task human subordinates, and who can expect from these systems some degree of flexible mission execution rather than having to teleoperate the vehicles. The current implementation of our system is running under the ModSAF (DIS) environment, allowing us to experiment with the capabilities of our techniques, and more broadly providing another suite of computer-generated forces (simulating UGVs) for training and evaluation

8. Acknowledgment

This research was sponsored by DARPA under contract DAAE-07-92-C-R012.

9. References

- [Cook et al] Cook, Diane; Gmytrasiewicz, Piotr; and Holder, Lawrence. Decision-Theoretic Cooperative Sensor Planning. To appear in IEEE Transactions on Pattern Analysis and Machine Intelligence.
- [Ingrand and Georgeff] Francois. F. Ingrand and Michael P. Georgeff. Managing Deliberation and Reasoning in Real-Time AI Systems. In Proceedings of the 1990 DARPA Workshop on Innovative Approaches to Planning, Scheduling, and Control, pages 284-291, Santa Diego, CA, Nov 1990.
- [Kluge et al] Kluge, Karl; Weymouth, Terry; and Smith, Ryan. Information Assimilation Research at the University of Michigan for the ARPA Unmanned Ground Vehicle Project. Sensor Fusion VII (SPIE vol. 2355), pp. 2-13, 1994.
- [Lee et al] Jaeho Lee, Marcus J. Huber, Edmund H. Durfee, and Patrick G. Kenny, UM-PRS: an implementation of the procedural reasoning system for multirobot applications. In Conference on Intelligent Robotics in Field, Factory, Service, and Space (CIRFFSS '94), pages 842-849, Huston, Texas, March 1994.
- [Huber et al] Marcus J. Huber, Jaeho Lee, Patrick G. Kenny, and Edmund H. Durfee, UM-PRS V3.0 Programmer and Reference Guide, University of Michigan AI Lab Document, 1994
- [Fedor] Christopher Fedor, TCX: Task Communications V8.6 Robotics Institute, Carnegie Mellon University, Pittsburg, PA, 1993
- [Musman et al] Musman, S.; Lehner, P.; Elsaesser, C. Sensor planning for moving targets. Sensor Fusion VII (SPIE vol, 2355), pp. 86-96, 1994.

- [Nii] Nii, H. Penny. Blackboard Systems: The Blackboard Model of Problem Solving and the Evolution of Blackboard Architectures. *AI Magazine* 7(2):38-53, Summer 1986.
- [Nii] Nii, H. Penny. Blackboard Systems: Blackboard Application Systems, Blackboard Systems From a Knowledge Engineering Perspective. *AI Magazine* 7(3):82-106.
- [Shaham] Shaham, Y. J. Two-dimensional recognition range model. *Infrared Systems and Components II* (SPIE vol. 890), pp. 92-94, 1988.
- [Stentz] Stentz, Anthony. The CODGER System for Mobile Robot Navigation. *Vision and Navigation: The Carnegie Mellon Navlab*. Charles E. Thorpe (editor), Kluwer Academic Publishers, 1990, Chapter 9.
- [Stentz] Stentz, Anthony. The NAVLAB System for Mobile Robot Navigation. PhD thesis, Carnegie Mellon University, 1990.
- [Van Brackle] Van Brackle, David; et al. Terrain Reasoning for Reconnaissance Planning in Polygonal Terrain. *Proceedings, Computer Generated Forces '93*.

10. Authors' Biographies

Patrick G. Kenny is a staff research programmer at the University of Michigan where he is currently working on the Unmanned Ground Vehicle project developing the system and architecture described in this paper. He received his Bachelor Degree from the University of Minnesota. His interests are in Autonomous Agent Architecture's and Minirobotics.

Edmund H. Durfee is an Associate Professor in the Department of Electrical Engineering and Computer Science at the University of Michigan, where he conducts research in multiagent systems, real-time intelligent control, and cooperative problem solving for applications ranging from interacting unmanned vehicles to supporting human collaboration. He received a PhD in Computer Science from the University of Massachusetts in 1987, and was named a Presidential Young Investigator in 1991.

Karl Kluge is an Assistant Research Scientist in the Department of Electrical Engineering and Computer Science at the University of Michigan. He received his PhD from Carnegie Mellon University, where he developed the YARF road follower. His interests are in mobile robots, vision, and intelligent vehicles.

An Architecture for Computer Generated Individual Combatants

Douglas A. Reece and Paul Kelly
Institute for Simulation and Training
3280 Progress Dr., Orlando, FL 32826
dreece@ist.ucf.edu

1. Abstract

The Institute for Simulation and Training is developing Computer Controlled Hostiles and Neutrals (CCH/N) to support an individual combatant trainer being developed by the USMC. The CCH/N system requires simulation of all aspects of human soldiers from physical actions to problem solving. At the physical level we have implemented visual and audio detection models, an action model, and "stubs" for aiming, wound effects, and fatigue. The control level implements continuous actions such as movement control. The action selection level determines the immediate focus of activity; this level contains the knowledge about what actions or subtasks are appropriate to accomplish a task in the current situation. Probabilistic selections at this level allow the system to simulate different behavior for different types of soldiers. Finally, the problem solving level performs long-term computations that cannot be completed in the time available to select the next action. These computations, such as route planning or terrain analysis, are started in response to requests from the action selection process. This paper describes the CCH/N entity architecture.

2. Introduction

2.1 TTES

IST is developing autonomous individual computer controlled hostiles and neutrals (CCH/N) to populate a virtual battlefield as part of the Team Target Engagement Simulator (TTES) project. This project, which is sponsored by the Naval Air Warfare Center Training Systems Division in Orlando, will develop a system to train small infantry units to fight in urban terrain. The users will initially be Marine Corps squads, but the system could potentially be expanded to other services, Special Forces operations, hostage rescue missions, etc. The system is designed to operate as a distributed virtual environment simulation using Distributed Interactive Simulation (DIS) protocols to link simulator nodes.

Simulating humans requires modeling all aspects of human behavior from physical actions to problem solving. In this paper we present the overall architecture of our artificial human agents and describe the models used at different behavior levels. We then focus on action selection behavior in more detail.

2.2 CCH/N Design Requirements and Constraints

Several requirements and goals directed the design of the CCH/Ns.

- In the distributed simulation environment, the public representation of the IC is currently constrained by available DIS entity state variables. Thus besides location and velocity, the only configuration state available is stance (standing, kneeling, prone) and weapon readiness (stowed, deployed, in firing position). The transitions between stances, gaits and weapon readiness states are animated at the image generator of the TTES trainee stations to avoid requiring entities to generate and broadcast body part information.
- The motions and state changes of the CCH/N must be realistic. Trainees do not just view the CCH/Ns from long range but can literally come face to face with them. For the CCH/Ns to look realistic, their speeds, accelerations, facing changes, and timings for stance and weapon readiness changes must be as accurate as possible.
- The domain of TTES is fireteam or squad level engagements in urban terrain. The scenarios usually put the CCHs on the defensive, although the engagements are more like meeting engagements than set battles. Meetings can be surprising and can occur at close range. Thus it has been more important to model individual perception and individual responses to new threats and changing situations than it has to encode unit tasks.

- The TTES CCH system is modeling different kinds of hostile soldiers—regulars, untrained militia, etc. Subject matter experts have provided opinions as to the differences in soldier behavior in terms of probabilities of different responses to certain situations. For example, the probability of engaging vs. seeking cover when first detecting a threat. Thus the CCH must not simply find the best action for the situation, but must identify a number of actions and choose one probabilistically. Most existing computer generated forces systems do not do this.
- The TTES CCH system simulates multiple entities on one host computer. The simulation is, of course, in real time. Thus the CCH architecture must provide a way to keep the CCHs responsive even when their reasoning process requires long computations.

3. Overall Architecture

The CCH/N system is divided into several distinct levels, as shown in Figure 1. At the bottom is the

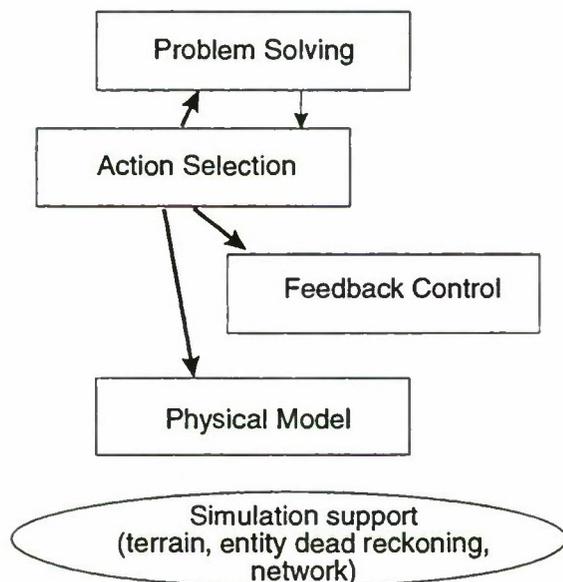


Figure 1. Levels of the CCH architecture.

physical level which describes the physical interaction of the CCH with the virtual world. Above that is the control level where behavior requiring continuous feedback control is implemented. In these two levels, continuous phenomena are simulated in time steps. The next level is the action selection level where knowledge is applied to select immediate actions. These actions

may either be physical actions or computation processes at the next higher level. Action selection takes place repeatedly in a “decision cycle” of about a second. The highest level is the problem solving level where long computations are performed.

3.1 Physical level

The physical level of the CCH contains all of the data and procedures that define the characteristics of the entity. The first aspect of the physical level that must be defined for ICs is a model of action. While there is no man-made hull to simulate, there is an analogous body to model with similar physical parameters—maximum speeds, accelerations, etc. Unlike most vehicles, humans can easily move in a direction other than the one they are facing. The human body also has a great many moving parts which potentially increase the complexity of its movement characteristics. Although in TTES body parts are not separately modeled, even the few DIS lifeform (soldier) states make the physical model fairly complex.

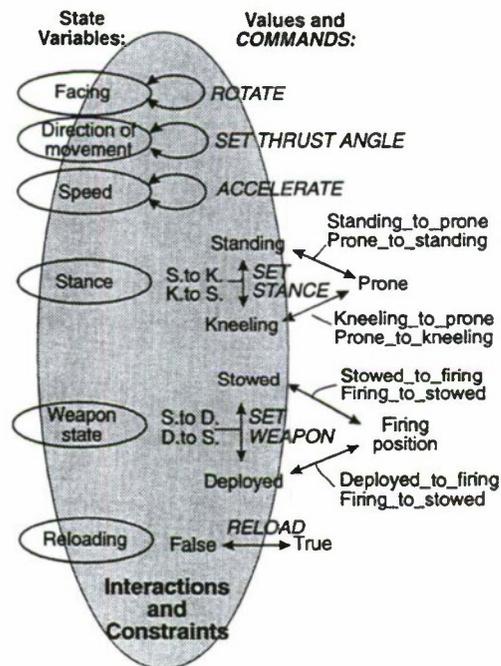


Figure 2. Interacting state variables in the CCH action model.

Figure 2 shows some of the necessary state variables and their values. The complexity arises from the interaction of the state variables. For example, what are the constraints on speed while holding the weapon in the firing position? Is it possible to fall prone while running? Is it possible to reload a weapon while rising to a standing position? The action model must specify how long it takes to perform various actions, and various combinations of actions. We are developing a model of human action (within DIS) that explicitly describes the state variable interactions, and are making much of it configurable with data files.

A second important aspect of the physical level is the perception model. Our perception model includes both vision and hearing. The vision component includes primary and peripheral fields of view with instant "pop-up" target detection in the peripheral field and search-based target identification in the primary field. The hearing component, which includes a simple sound generation and propagation model, allows CCHs to detect and sometimes identify other entities when they move or fire. Loud sounds mask softer sounds. A more detailed description of the perception model can be found in (Reece 1996).

We have incorporated weapon control into the CCH physical level. In a more detailed model of a soldier, aiming and firing a weapon would properly be part of the control level; however, we do not model the control of the position of body parts but treat aiming and firing as primitive physical actions. The result of firing is currently a hit probability rather than a ballistic round trajectory. Hit probability is based on a nominal maximum error radius for the soldier-weapon combination at 100 meters. The resulting hit area is compared with the visible target area projected at 100 meters. Target and firer motion and firer stance are factored in to the error radius. In the future we will also modify the error radius for aim time, wounds, suppression effects, and other factors that affect aim.

The fourth component of our physical level is fatigue. We currently use a simple fatigue model that reduces the CCH's "energy" as it moves; faster movement uses energy faster. Remaining stationary allows the CCH to regain energy. Reduced energy levels cause the CCH's movement capability to be reduced. This model is called from the dynamics routine in one place and could be replaced in the future by a higher fidelity model such as IUSS (Okeefe 1994).

The final aspect of the physical level is modeling wounds. Wound effects are difficult to simulate because they introduce further complications into the action model. We do not currently simulate wounds. This has not been an important requirement in TTES, partly because there are no visual effects to accompany wounds and partly because the human trainees suffer no wound effects.

3.2 Control level

The control level implements those behaviors that require continuous feedback control. In our CCH system this currently includes only movement activities. Tracking (facing) moving entities or avoiding moving obstacles requires constant sensing and motion correction to perform the task accurately. When movement along a path can be modified by moving entities or other disturbances, or when desired motion is not possible due to acceleration or turning limits, the path following task also requires feedback control to stay on track. Path following may include moving toward a point, movement along a road, movement along a wall, etc.

The control level in our CCH system is composed of a controller manager and a number of control modules. The controller manager receives commands from the level above and activates and connects the appropriate control modules together. Figure 3 shows these components implementing a simple route point following activity. The control modules take data from the physical level, the environment data base, and from other control modules via input "ports." The data produced by the modules are sent out of an output "port," which may be connected to another control module or to the physical level.

When the controller manager connects together a set of control modules, it forms a network through which data flows from sense input to control output. This concept has been used in the low levels of numerous autonomous agent designs (e.g. Brooks 1976 and Becket 1993). In our system the activated control modules are put on an execution list and executed in order every time step. The controller manager's implementation of the command functions must place the modules on the list in the correct order so that they are executed in the desired order. Generally, the desired order is from input to output so that there is no latency. The modules could be connected in loops, and the single pass execution would prevent race conditions from occurring.

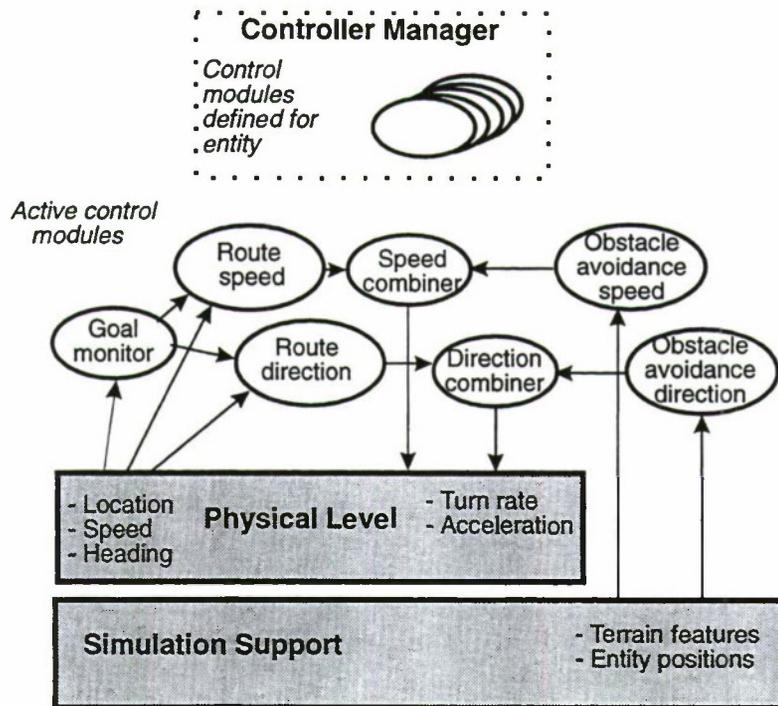


Figure 3. Controller manager and control modules for simple route following with obstacle avoidance.

3.3 Action selection

The action selection level is the center of cognitive activity for the CCH. Decisions at this level initiate all physical and problem solving activities. Outputs can go directly to the physical level to perform actions, to the control level to start physical activities, or to the problem solving level to start long computations. This system organization is similar to other that of other intelligent agent architectures such as (Becket 1993, Gat 1992, Mettala 1992, and Reece 1993).

The action selection layer runs periodically like the control modules but with a longer period. In addition, significant events can trigger action selection to run before its scheduled time—for example, near misses from weapons or new, nearby threat sightings. The action selection computation is intended to be fast so that it (as well as control, physical model, and computations for other entities) may be run frequently and without situation-dependent delays. Long computations are performed at the problem solving level so that the CCH always remains responsive even while thinking. It is desirable to set an upper bound on action selection computation time in order to guarantee real time responsiveness in all situations; however, we cannot

yet guarantee an upper bound on the CCH action selection process because the CCH's simulated perception requires some searching of a database instead of indexing directly to interesting objects (see Section 4.2 below).

3.4 Problem solving

The top level performs mental actions that take a long or unbounded amount of time to finish. Typically these are route planning, terrain analysis or mission planning tasks. The expense of the tasks comes from a search through a large space of alternatives or from expensive numerical calculations, or both.

The functions in the problem solving level are run in separate processes from the action selection and other levels. The CCH system multitasks between the entity's processes (and between processes of different entities) using cooperative multitasking; i.e. the processes have to give up control of the processor voluntarily. Functions at the problem solving level are written to allow cooperative multitasking. Processes implementing the lower levels can continue to run while long problems are being solved; thus the entity stays responsive to the environment. The problem solving processes may be run at a lower priority than

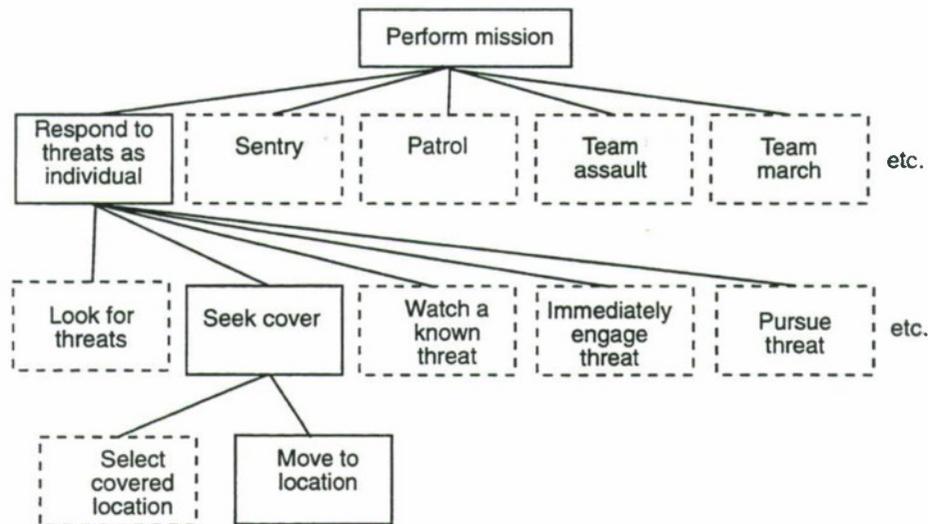


Figure 4. Partial hierarchy of tasks and subtasks known to a CCH. The solid boxes show an example of what subtasks a CCH might be performing at a moment in time.

others to ensure that the entity moves and reacts properly even under higher processor load.

When the action selection level determines that a long computation is needed, it starts a problem solving process and remembers that it did so. While the computation is being undertaken, the action selection level chooses the appropriate action given its lack of information. Action selection does not restart the process every time it runs, but may update data provided to the process. If this invalidates the problem process' work so far, it might have to start again. If the problem solving process is written in such a way that it can provide partial results (e.g., the first part of a route plan) before it is done, the action selection level can take advantage of these results.

4. Action Selection

The action selection level consists of a hierarchy of subtasks that allow the entity to decompose its tasks into primitive activities; knowledge of what subtasks can accomplish a task in a given situation; and an inference engine that applies the knowledge to start subtasks and activities.

4.1 Tasks

The CCH system encodes domain-specific knowledge about action in an object called a *task*. Each task contains a list of subtasks which can help, in some situation, the entity accomplish the task. In addition the task has a set of rules that describe how to select subtasks. The subtasks can themselves describe

subtasks, so that the entity's knowledge about a task is a hierarchy of tasks and possible subtasks. A number of intelligent agent designs described in the literature use task hierarchies, but they often have a strong flavor of finite state machines that follow limited sequences of states (e.g. Ahmad 1994, Calder 1993). Our task hierarchy is more like (Tambe 1995) in that it is intended to provide a set of subtasks that may be freely chosen to best accomplish the task in the current situation.

At the bottom of this hierarchy, tasks may start physical or mental actions. In terms of our CCH architecture, the tasks may request an action from the physical level, start a controlled activity at the control level, or start a mental process at the problem solving level. The current focus of attention of the CCH at any time is describe by the stack listing the active task, subtask, sub-subtask, etc. Figure 4 shows part of the task hierarchy of a CCH with the current state indicated; this entity is currently seeking cover in response to a new threat.

4.2 Rules For Selecting Subtasks

Knowledge about how to accomplish tasks is encoded in *rules*. Rules examine facts in the CCH's "memory" to assess the internal and external situation and then either propose subtasks to start or reject subtasks. A rule may, for example, reject *Immediately_Engage* if the entity's weapon is not loaded or if the entity is out of ammunition. The proposals have a coarse priority assigned to them so that important or default proposals may be indicated.

```

1. if
  (Received_close_fire)
  (Range_to_highest_threat >
   CLOSE_QUARTERS_RANGE)
  then
    (propose_action
     ((name Seek_cover)
      (priority VERY_HIGH)))
2. if
  (Threat_with_LOS)
  (Range_to_closest_threat <
   CLOSE_QUARTERS_RANGE)
  (Weapon_loaded)
  then
    (propose_action
     ((name Immediate_engage)
      (priority HIGH)))
3. if
  (Threat_with_LOS)
  (Have_ammo)
  then
    (propose_action
     ((name Immediate_engage)
      (priority MEDIUM)
      (weight 20)))
4. if
  (Threat_with_LOS)
  then
    (propose_action
     ((name Seek_cover)
      (priority MEDIUM)
      (weight 80)))

```

Figure 5. Sample rules for selecting subtask of Respond_to_threat_as_individual task.

For example, a suppression response to a nearby bullet impact might have a high priority, while a proposal to look around might be the default action. Figure 5 gives some sample rules for selecting actions in the Respond_to_threats_as_individual task from Figure 4. In the future we plan to allow rules to specify preferences for one subtask over another as a more situation dependent way of expressing the importance of subtasks.

Rules provide a “probability weight” with each proposal. This weight is intended to indicate the relative probability of selecting the given subtask from among the subtasks proposed with the same priority. This mechanism is necessary to implement behavior variations between different types of CCHs. Subject matter experts have reported, in studies for TTES, the probability distribution of different behaviors in different situations for different types of soldiers (Lind 1995). Our action selection

mechanism is thus required not to find the best subtask to accomplish a task, but to propose a set of potential subtasks and select among them randomly. This requirement is generally not addressed by intelligent agent architectures in the literature.

The CCH memory for action selection rules is an object with slots pre-assigned for all facts that are used by all of the rules. These facts are objects with a value and an evaluation function. The evaluation function can call simulator access functions to get terrain or other entity information, or use “perception” functions to access state variables in its own entity or otherwise find data.

Our rule mechanism does not have the expressibility of traditional rule-based inference systems. In particular, the rules have very limited use of variables. Generally, facts and expressions in rules have only one attribute or parameter. There are two reasons that we have chosen this design. First, the inference mechanism is simple, small, and fast; it was easy to integrate the mechanism as a module into the existing body of simulation code. The CCH memory described above, combined with the simple rules, avoids the need for memory management support such as garbage collection. The rule based action selection is run on a different CCH memory and potentially a different rule set for each CCH simulated in the program.

The second reason for designing our rule system this way is to try to make the action selection process more directed by a focus of attention. A traditional rule system must examine all combinations of facts that might match the logical expressions in the left hand sides of the rules. Consider the problem of finding the closest visible threat entity; a logical rule would have to describe this entity A as

$$\begin{aligned}
& Me(E) \wedge \\
& Threat(A) \\
& Threat(B) \wedge \\
& Visible(A, E) \wedge \\
& \forall B, Range(A) < Range(B)
\end{aligned}$$

There is no explicit control over how the system goes through the entities checking visibility (a relatively expensive operation) and threat and comparing ranges pairwise. Our goal is to create the “perception” functions mentioned above so that they identify facts relevant to the entity as directly as possible, and then to write rules using these salient facts. For example,

our rule would not use the above expressions in the left hand side, but would refer to "the_closest_visible_threat_entity." This approach was described by (Agré, 1987). Their Pengi system used action proposers in a combinatorial decision whose inputs were "indexed" to the entity. The inputs were generated by "visual routines" that found important inputs directly. In our case we cannot always find inputs without searching short lists of entities or terrain features, but we at least have explicit control over the search mechanism.

4.3 Action Selection Process

The action selection process is run repeatedly about once a second. It can also be run immediately in response to a significant event such as an activity completion, a new sighting, or a fire event. Each time action selection is run, the top level task is examined. Its rules are evaluated and a subtask is selected. If it is the same as the currently active subtask, the action selection process drops down the task stack to this subtask and repeats the task evaluation process. If a new task was selected, the old subtask and, recursively, its subtasks, are stopped; the new subtask is then started and its rules evaluated immediately.

When the rules of a task are evaluated, they create a list of proposed subtasks with priorities and probability weights. An arbitration function throws away all but the highest priority tasks. If the current task has been proposed again, it is selected immediately. This prevents the CCH from randomly selecting a new task each cycle and "dithering" between tasks. If the current task has not been proposed, the weights of the proposed subtasks are normalized to the total weight of those proposed and a random number is generated to select a subtask.

4.4 Minimizing the Computation of Facts

One persistent challenge of developing computer generated forces is computing quickly the information that is immediately available to humans. For example, a human can look at a region of terrain and immediately identify a rise behind which he can hide from a threat; for a computer to find this location, it generally has to sample points on the terrain and make intervisibility tests. Many of the rules that we might create to control CCH behavior could test such facts in their left-hand-sides. For example, a rule

might test something about a nearby cover location, thus requiring the above intervisibility computations.

Given that many facts are expensive to compute, it is highly desirable to avoid doing this unless it is necessary. We have implemented three simple mechanisms to help avoid unnecessary computation of facts. The first is simply a lazy evaluation procedure; no facts evaluation functions are run until needed to evaluate a rule. The second mechanism is a cache. Fact objects contain a flag that indicates whether the fact has been updated during this decision cycle. When a rule requests a fact, the flag is consulted to see if the evaluation function must run. When the fact is evaluated, the result is stored for the next time and the flag is set. Thus fact evaluation functions are only run once even if more than one rule uses the same fact.

The third mechanism we use to avoid fact evaluation is rule ordering and incremental arbitration. We deliberately avoid the traditional forward-chaining inference mechanism because it must check all facts to see if any rules can fire. The rules are grouped according to priority and only one group is activated at a time. The groups are activated from highest to lowest priority. Rather than waiting until all groups have been considered, the arbitration mechanism is invoked after each group is evaluated. Thus if a rule proposes a subtask with a high priority, none of the rules that propose lower priority subtasks need to be considered—i.e., need to evaluate their facts. Currently the rules are grouped by hand, but this would be straightforward to do automatically.

5. Conclusion

We have developed an architecture for computer controlled individual combatants that defines modeling levels for physical characteristics, activity controlled by feedback, responsive action selection, and problem solving. Within this architecture we have developed new models for ICs at several levels. At the action selection level, we have defined domain knowledge representation using a task hierarchy and rules to select subtasks in different situations. Our action selection mechanism can select behavior probabilistically, and avoids unnecessary evaluation of computationally expensive facts in the rules.

6. Acknowledgement

This work is been supported by contract N61339-94-C-0006 from the Naval Air Warfare Center Training Systems Division.

7. References

- Agre, P. and Chapman, D. (1987) "Pengi: An Implementation of a Theory of Activity." *Proceedings of AAAI-87*, pp 268-272.
- Ahmad, O. *et al* (1994) "Hierarchical, Concurrent State Machines for Behavior Modeling and Scenario Control", in *Proceedings of the Fifth Annual Conference on AI, Simulation, and Planning in High Autonomy Systems*, IEEE Computer Society Press.
- Becket, W. and Badler, N. (1993) "Integrated Behavioral Agent Architecture", In *Proceedings of the Third Conference on Computer Generated Forces and Behavioral Representation*. University of Central Florida.
- Brooks, R. (1986) "A Robust Layerd Control System for a Mobile Robot," *IEEE Journal of Robotics and Automation*, Vol RA-2, No. 1.
- Calder, R., Smith, J., Mar, J. and Ceranowicz, A. (1993) "ModSAF Behavior Simulation and Control", in *Proceedings of the Third Conference on Computer Generated Forces and Behavioral Representation*. University of Central Florida.
- Gat, E. (1992) "Integrating Planning and Reacting in a Heterogeneous Asynchronous Architecture for Controlling Real-World Mobile Robots," In *Proceedings of AAAI-92*, pp. 809-815.
- Lind, J.(1995) *Behavior Representation for the Team Tactical Engagement Simulator (TTES)*, Naval Air Warfare Center Weapons Division.
- Mettala, E. (1992) "The OSD Tactical Unmanned Ground Vehicle Program," In *Proceedings of the DARPA Image Understanding Workshop*.
- O'Keefe, J. (1994) *Factors Governing Dismounted Human Movement in a Synthetic Environment*. U.S. Army Natick Research, Development and Engineering Center.
- Reece, D. (1993) "Execution Control for CPU-Sharing Agents," In *Proceedings of the Third Conference on Computer Generated Forces and Behavioral Representation*. University of Central Florida.
- Reece, D. and R. Wirthlin, (1996). "Detection Models for Computer Generated Individual Combatants", In *Proceedings of the Sixth Conference on Computer Generated Forces and Behavioral Representation*, University of Central Florida.

Tambe, M. *et al* (1995) "Intelligent Agents for Interactive Simulation Environments," *AI Magazine*, Spring 1995.

8. Authors' Biographies

Douglas A. Reece is a Computer Scientist at the Institute for Simulation and Training. He is the Principal Investigator of the TTES Computer Controlled Hostiles project. His research interests are in artificial intelligence, specifically intelligent agent design and computer vision. He has a Ph.D. in Computer Science from Carnegie Mellon University and B.S. and M.S. degrees in Electrical Engineering from Case Western Reserve University.

Paul Kelly is a graduate student in Computer Science at the University of Central Florida. He has worked on the TTES CCH project for two years and is using it to study and develop action selection methods for intelligent soldier agents. He received a B.S. degree in Computer Science from the University of Central Florida.

Session 5b: VV&A

Paz, U. S. Army, STRICOM

Marshall, U. S. Army, STRICOM

Mastroianni, U. S. Army Natick RD&E Center

Mullis, U. S. Army, TRAC-WSMR



ModSAF Credibility

Benjamin D. Paz
U.S. Army Simulation Training and Instrumentation Command
Orlando, Florida 32826-3276
pazb@stricom.army.mil

Irwin L. Hudson
NATIONS, Inc.
Orlando, Florida 32826
hudsoni@stricom.army.mil

1. Abstract

The precise alignment of the Software Development Process to the implementation of Verification and Validation (V&V) has been proven to be critical in determining the credibility of models and simulations (M&S). Since ModSAF is an M&S which has unanticipated requirements to meet the needs of various programs, its development strategy has to be unique and versatile. There has to be a standard process in place to enable the community to develop new and robust capabilities for integration into the ModSAF baseline. In order to ensure credibility, this development process must be paired with an equally standardized V&V plan. Hence, the ultimate goal is to leverage from various programs and produce credible releases of ModSAF.

2. Introduction

2.1 ModSAF Overview

Modular Semi-Automated Forces (ModSAF) is a widely used Computer Generated Force simulation for use in Distributed Interactive Simulation exercises. ModSAF provides simulated forces and environmental effects on the virtual battlefield. Only elements which are externally visible or significant are simulated. A heuristic approach is used to simulate behaviors. This approach lends itself well to computational efficiency but does not provide for complete automation. An operator is required to plan the orders for units and intervene in situations where the automated behavior lacks an appropriate response.

2.2 Background

ModSAF development began in the spring of 1992 with the objective of providing an open, modular architecture which researchers could enhance and

extend with ease compared to the previous SAF systems. In mid 1993 the Battlefield Distributed Simulation - Development (BDS-D) program began the process of identifying legacy capabilities to integrate with ModSAF. These capabilities were often no more than demonstration quality in order to provide proof of concept for SAF development.

Early in 1994 the Anti-Armor Advanced Technology Demonstration (A2 ATD) embarked on the effort for Verification, Validation, and Accreditation of the BDS-D environment. During the course of the A2 ATD, weaknesses of the software development and configuration management processes became apparent as obtaining and developing data and then controlling the data in the baseline were very complex to manage.

In 1995, the Computer Generated Forces Assessment Working Group (CGFAWG) was formed for the Deputy Under Secretary for the Army for Operations Research (DUSA-OR) to conduct a study of CGF for use in DIS. The study identified ModSAF as being capable to support Distributed Interactive Simulation (DIS) needs for the Advanced Concepts and Requirements (ACR), Research Development and Acquisition (RDA), and Training Exercises and Military Operations (TEMO) domains. This finding resulted in the perception across the Army community that ModSAF actually possesses these capabilities (not just the potential). However, ModSAF did not possess all the capabilities (and certainly not the domain specific V&V).

By late 1995 the results of the CGFAWG (Brooks et al. 1996) spread throughout the Department of the Army. Throughout 1995 ModSAF was in a transitional phase from the BDS-D program manager to the Program Manager for Distributed Interactive

Simulation (PM DIS). Many new ModSAF users were unaware of the origin of the ModSAF program ignoring the transition status and expected a operational system with the V&V that robust Configuration Management (CM) supports for exercises or training scenarios. Under the technology based BDS-D Program, although capabilities were developed from military doctrine, V&V traceability was not sustained due to the lack of CM.

Today ModSAF Simulation of military doctrine and platforms does not perform as expected and as a result user confidence in the system has fallen.

2.3 Objective

The DUSA-OR requires PM DIS to develop and implement a ModSAF CM process which institutionalizes ModSAF V&V as implemented in the A2 ATD and the BDS-D program (Hollis 1995). A standard configuration managed process aligning software development and verification and validation is needed to provide the required traceability.

2.4 Mission Statement

The objective for this paper is to inform the ModSAF user and developer communities on the enhanced development process. This new process provides CM for the entire system, links V&V to software development, and provides a plan for V&V of legacy capabilities.

3. Configuration Management

An extended software development approach now exists which includes V&V from the onset. A robust CM process must be in place to support the development (PM DIS 1996a). Existing and future capabilities for ModSAF must conform to this

process and must be published in a manner readily accessible by all users and developers.

3.1 Development Process

To begin the process of making ModSAF more credible, PM DIS established a Configuration Control Board (CCB) for the ModSAF program (PM DIS 1996c). The CCB is chartered to systematically control changes and maintain the integrity and traceability of the functional requirements. Under the umbrella of Integrated Process and Product Management (IPPM), two working arms of the CCB have been established: the Integrated Product Team (IPT) and the Integrated Development Team (IDT). The teams include membership and participation from government and industry. The IPT conducts the management of program activities while the IDT performs technical assessments for developments and problems. The hierarchy of the CCB structure is represented in Figure 1 below.

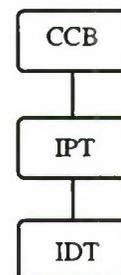


Figure 1 : CCB Structure

The IPT meets on a regular basis approximately once every two months. The IDT is organized into a series of working groups which report progress to the IPT. Methods for improving a particular process may invoke a temporary IDT working group while the process of reviewing

requirements are verified against system requirements. This level of V&V up front is designed to reduce the risk of developing the wrong functionality.

The subsequent phases of software development are typical to what has been performed since ModSAF version 1.0. Based on the KA/KE products, the design phase begins. Once a design is complete the V&V team will have planned what elements of the design to review such as user interface, timing and sizing, and training issues. A design report will be produced.

Of course, following design, programming begins and culminates in new code. The V&V team will review the code for integrity, compare the code to the design and evaluate interfaces as necessary. A code review report will be produced. Following completion of a capability, tests will be developed against the software requirements.

The phases of designing, programming and testing are typically incremental which means capability will be ready for integration prior to completion of the entire functional area under development. Incremental integrations will provide the user with the opportunity to check the new functionality and provide feedback. This feedback should be positive in that the up front V&V will have minimized invalid KA and KE. The feedback will provide input for future enhancements in place of corrections and fixes.

Immediately following the integration of new capability regression tests are performed (Courtemanche and Ceranowicz 1995). When the baseline planned freeze date is reached all the incremental integrations of capabilities and its associated regression test is complete, the release testing process begins. In Figure 2, the inner curve of the spiral represents the release testing process which starts with the frozen baseline.

During this time, release and acceptance testing are performed. The Beta version is released to user sites and other developer sites. User sites are responsible to find problems based on ModSAF capabilities and submit Problem/Trouble Reports (PTR) in their functional areas. Developers will submit PTRs based on stressing the system to find anomalies and problems in their functional area. The IDT is responsible for tracking and correcting PTRs.

Government representatives to the IDT can close PTRs that only require technical expertise. Those PTRs requiring tactical expertise will be referred to the appropriate user site for closure.

3.2 Proliferation

A key element is the exposure of this process to the entire community. One method for achieving this is to use the world wide web. The web site is located at <http://www.modsaf.org>. Access to the web site is via user and developer accounts. To obtain an account a ModSAF Distribution Agreement must be in place with PM DIS. CM maintains the ModSAF program status on the web. Information to be published includes the status of PTRs, changes and additions to software libraries, development schedules, future plans, and user help. This system will provide a means for users and developers to enter new PTRs and obtain help from the Frequently Asked Questions pages.

The ModSAF Configuration Status Accounting (CSA) system will be a means for users and developers to check the development process. References to documentation including V&V reports will be available on line. The traceability from requirements through test will be available.

Documentation developed for the system will be written according to the new ModSAF Software Development Plan (SDP). This new documentation will support traceability by mapping test procedures back to the initial system requirements. The SDP will be made available for the user community to understand what is required for integrating externally developed capabilities and how to successfully accomplish it.

As new configuration managed versions of ModSAF are released the spiral process of obtaining feedback from the ModSAF user and developer community. This feedback is tied into the process not only with PTRs but also with user enhancement requests which can be entered into the web system. After review by the IDT and IPT, the requests will be presented to the CCB

3.3 CM Database

The documentation and code of the ModSAF system will be stored in the CM database. This database is

not accessible through the web. The database will possess the links between the various documents and also to code. Change information will be stored as it is approved.

3.4 Legacy Capabilities

Legacy capabilities in ModSAF typically did not have CM to a satisfactory level for V&V. In performance of a ModSAF development effort legacy capabilities may be impacted. In this case, this extended development process will V&V the legacy capabilities in building the new functionality.

The remainder of the ModSAF system that has not been developed with V&V will require a plan to be completed. The plan must include transforming the legacy documentation into the new format for V&V review of the documents and code. As part of the transformation, a reverse engineering effort on code may reveal dead code which will subsequently be deleted. Other code may require rework. For each case involving rework the CCB will make the decision on repairing an implementation or removing its invalid representation.

3.5 Future Objectives

ModSAF is a system built with legacy components and structures. The architecture needs enhancements to ease maintainability. For example, standardizing data formats and adding tools for novice users to be able to enter and change military information, including tactics, would greatly enhance system usability.

Creation of mission statements for the ACR, RDA and TEMO domains will determine how ModSAF must be adapted for each domain. The adaptations needed will cause further changes to maximize leveraging between the domains. The differences between the domain requirements may include substitution of new physical models with greater detail while the behavioral characteristics remain the same. All of these changes will lead to more confidence in ModSAF by the user community.

4. Verification and Validation

4.1 Objectives

The V&V methodology for ModSAF is derived with this premise in mind: The main purpose for conducting V&V for ModSAF is to provide an "Accrediting" authority with sufficient information for determining whether ModSAF adheres to its intended purpose and use, as prescribed by the given requirements.

The Verification process applies to the functionality of the software as it relates to the requirements, conceptual model, and design. The Verification process determines whether or not the software development functions according to the requirements and accurately represents the user's conceptual descriptions and specifications.

The Validation process determines the manner and degree to which the behavior of a M&S is an accurate representation of the real world. Validation addresses the credibility of the requirements, by analyzing the conceptual model and testing the behaviors and physical models of the M&S.

V&V provides several benefits to software development. When implemented properly, it identifies deficiencies early in the development cycle, which allows for early corrective actions that reduce impacts to the program's cost and schedule. It also provides technical monitoring of developer's efforts, which consequently helps to ensure that the development complies with requirements.

V&V gives the user a concurrent look into the performance of the software. It provides necessary input which improves the quality of the software documentation (i.e. data, models, requirements, design, code, and test material). Furthermore, Verification, Validation, and Accreditation is required for all M&S products that are developed under the Army Model and Simulation Management Program (AMSMP). Guidance is provided by Army Regulation 5-11 and the Department of the Army Pamphlet (DA 1992, 1993)

4.2 ModSAF V&V

The basis for ModSAF V&V will be centered around a team approach to ensure a successful product. The ModSAF V&V Team shall be responsible for coordinating and conducting the required V&V activities on the specific components that comprise each individual ModSAF development effort. The team shall include various organizational participants

in the V&V field for the components or modules being tested, along with independent evaluators.

As mentioned earlier in this paper a specific V&V plan will be prepared for each ModSAF development effort by the assigned V&V Team. This plan will be tailored from the V&V Methodology for ModSAF Software Production (PM DIS 1996d) in accordance to the ModSAF Software Development Plan.

Verification analysis will be performed on the Software Requirements Specification to ensure that the developers requirements map back to initial system requirements produced by the user. Software design will begin only after the conceptual models for the development effort have been validated by government representation. A V&V report will be generated to show the results of the verification and validation analysis performed on the SRS and Conceptual Model. There will also be a Verification report produced for the analysis performed on the software design. Another Verification report will be generated for the analysis performed on the code and test documentation.

Individual Validation reports will be generated for the testing performed on the Behaviors and Physical Models of ModSAF. After final testing, the V&V Team will perform an overall assessment of the final product. A final V&V report containing results, conclusions and recommendations will be generated and made available to the accrediting authority.

The above process plays a critical role in producing a "credible" ModSAF development product.

5. Summary

This paper summarizes the current status of V&V in the ModSAF Program Plan, ModSAF Configuration Plan, ModSAF Software Development Plan, and the Verification and Validation Methodology for ModSAF Software Production. The overall guidance for this paper is from the ModSAF Program Plan. The content of this paper is abstracted from the these plans.

Throughout this paper the emphasis is on maintenance of V&V traceability and not the individual activities supporting V&V. CM is the key element that links software development to V&V.

6. Acknowledgments

The ModSAF Program was supported by STRICOM's Technology Base Program Manager, Mr. Stanley Goodman, from 1992 to 1996. Now the program is supported by PM DIS, Colonel James Etchechury and run by the ModSAF Project Director, Major Reba Lyons. Mr. Goodman is the new deputy for PM DIS.

7. References

- Brooks, W., Dymond, P., and Sandmeyer R. (1996) "Computer Generated Forces (CGF) Assessment", *Special Publication No. 72*. AMSAA, 190 Pages.
- Courtemanche, A.J., and Ceranowicz, A. (1995). "ModSAF Development Status", Proceedings of the Fifth Conference on Computer Generated Forces and Behavioral Representation, Orlando, FL: Institute for Simulation and Training. pp. 3-13.
- DA, (1992) "Army Model and Simulation Management Program", *AR 5-11*, Headquarters Department of the Army (DA).
- DA, (1993) "Verification, Validation, and Accreditation of Army Models and Simulations", *Pamphlet 5-11*, Headquarters DA.
- Hollis, W.W. (1995). "Computer Generated Forces (CGF) Assessment", *Memorandum SAUS-OR 28 November 1995*. Deputy Undersecretary of the Army for Operations Research.
- PM DIS (1996a) "ModSAF Configuration Management Plan", STRICOM
- PM DIS (1996b) "ModSAF Software Development Plan", STRICOM
- PM DIS (1996c) "ModSAF Program Plan", STRICOM
- PM DIS (1996d) "Validation & Verification Methodology for ModSAF Software Production", STRICOM

8. Bibliographies

Benjamin D. Paz is a Systems Engineer for the U.S. Army Simulation Training and Instrumentation Command (STRICOM). At STRICOM, Mr. Paz is the Lead Systems Engineer for the ModSAF simulation. He holds Master and Bachelor degrees in Computer Engineering from the University of Louisville, Speed Scientific School. His interests include CGF simulation and software engineering.

Irwin L. Hudson is a Software Engineer with NATIONS, Inc. Mr. Hudson is supporting STRICOM's efforts in ModSAF V&V. He has a Bachelor's of Engineering degree. His interests include V&V and human factors engineering.



SAF and Manned Simulators Correlation Issues in CCTT

Henry Marshall
Simulation Training and Instrumentation Command
marshall@stricom.army.mil

Edward V. Chandler, Brian R. McEnany
Science Applications International Corporation
echandle@greatwall.cctt.com
brian_mcenany@cpqm.saic.com

John G. Thomas
U.S. Army Materiel Systems Analysis Activity
jgthomas@arl.mil

1. Abstract

This paper attempts to define issues relating to the correlation between the Semi Automated Forces (SAF) and the manned SIMulators (SIMs) in the Close Combat Tactical Trainer (CCTT) Distributed Interactive Simulation (DIS) domain. The correlation or what is sometimes termed 'Fair Fight' is critical to the acceptance of the CCTT system as an accredited training simulation system. If left uncorrected, many of these issues will create negative training environment for the crews of the simulators.

2. Introduction

The development of CCTT mandates the use of validated data and models which have been provided by Army Materiel Systems Analysis Activity (AMSAA) and other sources. Some data and model sets were not available from approved providers and had to be synthesized during the development. These models have been developed based on real world empirical and engineering data and typically have problems correlating to the limitations and performance of current visual system designs. The CCTT visual system is constrained by a 4000 meter magnified range and a 2400 meter unity vision

Correlation issues have been identified in a wide number of areas including; damage assessment, mobility/trafficability, behaviors, rate of fire, delivery accuracy, and target acquisition. For example, during the User exercises the most common concern expressed by the SIM operators was how easily the Opposing FORCE (OPFOR) SAF acquired and engaged the SIMs, despite the fact they were moving and/or the simulators were positioned in a well concealed area. The target acquisition problem is the most difficult and critical fair fight issue. As part of

our discussion, we will explore the ACQUIRE model which is the validated target acquisition methodology, its limitations, rationale, as well as limitations of the CCTT simulators visual environment and behavioral aspects of acquiring and engaging targets.

Other correlation issues such as damage assessment and mobility stem from, providing a maximum fidelity model for the simulators, and a limited fidelity model for the SAF. This strategy reduces computational and data storage overhead for SAF processors while ideally allowing minimal correlation differences.

The paper will also show current testing and action plans to address these issues in the development of CCTT. The contractual requirements to correlate SAF and modules are part of Baseline Change Request (BCR) 174. The purpose of BCR 174 stemmed from experience gained in observing differences between SIMs and SAF entities in SIMulation NETwork (SIMNET) and Modular Semi-Automated Forces (ModSAF). A four month investigative effort began last year to examine what legacy differences existed, what was the current state of similar development areas in CCTT, determine if specific experiments to test the significance of differences were needed and finally, were modifications to ongoing CCTT development warranted to correct any correlation issues. The results of this investigation provided several recommendations, of which, performing experiments on target acquisition capabilities between SAF and SIMs was the most significant. Currently CCTT is in the process of integrating and testing functionality in four separate blocks designed to grow the overall system functionality. This effort will be going on through the end of this year. An important part of each block will be to perform new experiments and integrate new data and findings of

the BCR 174 effort. Currently, the CCTT system has matured to the point where experiments to derive useful data can occur. However, as of the writing of this paper, the progress of BCR 174 is limited to requirements definition and preliminary testing. Detailed implementation finding will be discussed in future papers and be added to future versions of the AMSAA CCTT Data Compendium.

3. Correlation Issues

3.1 Damage Assessment

The principal determinants of damage assessment are combat damage, stochastic failures, and deterministic failures. The damage assessment process will identify the component or system that was damaged or failed, the organization that can repair the damage or failure (either crew or Unit Maintenance Collection Point [UMCP]), and the time to repair the damage or failure.

Under combat damage, once it has been determined that a target has been hit, an assessment of whether the target is killed needs to be made. Probabilities of kill given a hit are based on impact conditions on the target, such as damage, location of hit on the vehicle, range from the firer to the target, attack angle, and type of ammunition used. Kill criteria are as follows: mobility (M) kill - an armored vehicle suffers M kill if it is incapable of executing controlled movement within a short time after being hit and is not repairable by the crew; firepower (F) kill - an armored vehicle suffers F kill if it becomes incapable of delivering controlled fire within a short time after being hit and is not repairable by the crew; M or F kill - an armored vehicle suffers M or F kill if it suffers a mobility or firepower kill; and catastrophic (K) kill - an armored vehicle suffers a K kill if it is destroyed or suffers both mobility and firepower kills and is not economically repairable.

A stochastic failure occurs when the vehicle or equipment fails on its own, not through crew error or combat damage. The frequency of failure is determined by the Mean Time Between Failure (MTBF) for the particular vehicle, based on AMSAA provided reliability data. The current design does not support OPFOR SAF stochastic failure. This is desired because there is no training value in having a OPFOR operator, which is a contractor, recover a OPFOR vehicle that failed in a stochastic manner.

A deterministic failure is a failure that occurs due to resource depletion or improper action. These type failures include mismanagement of fuel and ammunition, collisions, thrown tracks from high speed,

and warnings ignored by the crew. SAF entities avoid situations that lead to deterministic failures, with exception of running out of fuel and ammo.

For combat damage, SIMs and SAF use essentially the same data provided by AMSAA. However, SIMs use component level damage data; whereas SAF uses a distribution to ascertain mobility kills, firepower kills, and catastrophic kills. The components that SAF can damage is limited to; engine, tracks, weapons, turret, sensors, fuel transfer pump, computers and transmission. For stochastic failures, SIMs use component level failure data as provided by AMSAA; whereas SAF uses mobility, firepower, and electrical/sensor subsystem failure data.

AMSAA did not provide deterministic failure data for SIMs or SAF. Data was generated by Integrated Development Team (IDT) for SIMs and SAF. SIMs will simulate more deterministic type failures than SAF and will include starter motor and laser range finder failures, rollover, drowning, and thrown tracks; whereas SAF will only simulate thrown tracks, collision, out of fuel and out of ammo. Note, in the current baseline, SAF will receive collision damage where as the SIMs react to collisions dynamically only. The SIMs that drowns will receive engine damage and get stuck. The SAF entity will get stuck and halt.

Findings and Concerns - The main difference in all damage assessment between the SAF and SIMs, is that SIMs uses component level failure data while SAF uses mobility, firepower, and electrical/sensor subsystem failure data. As a result, SIMs can randomly degrade some mobility and firepower failures and continue its mission while SAF either stops moving or stops firing. The current methodology used by SAF does not lend itself to modeling degradation in mobility or firepower. Since a typical SAF Computer Generated Forces (CGF) processor is targeted to control upwards of fifty entities, verses one for the SIMs. SAF can not support the higher fidelity component and subcomponent modeling the SIMs can.

For deterministic failures, the difference between SIMs and SAF is the number of type failures that can occur. The ability of SAF to receive collision damage which the SIMs do not may produce a disadvantage for SAF.

Despite the data differences in the levels of damage and failure and the man-in-the-loop, damage assessment is not a major problem for SIMs and SAF. Currently the major problem in testing, is that some vehicle types are very difficult to damage with certain types of ammunition providing non logical results. The SAF

damage assessment method does not provide for cumulative damage effects, making situations where a vehicle will be hit numerous times and only receive numerous mobility kills.

3.2 Mobility

Differences exist between SAF and SIMs in terms of hull motion dynamics. SIMs use manufacturer's engineering data combined with algorithms based on dynamic Newtonian models. When combined with crew actions, they achieve realistic, interactive movement across the terrain. The SAF system is based on Newtonian models, enhanced legacy algorithms, data-driven models, or modified SIMs algorithms. They provide generically realistic SAF behaviors. Validated vehicle mobility data from Waterways Experiment Station (WES) is used to approximate the interactions between vehicle and terrain for both SAF and to test and validate the SIMS overall automotive performance.

SIMs have a high level of detail and fidelity. For the SIMs, each roadwheel is independently modeled. There is full engine and drivetrain simulation that provides the crew with all needed visual, aural, and motion cues. WES-supplied terrain characteristic coefficients are combined with vehicle system simulation and operator input to result in dynamically varying behavior. The level of fidelity approximates that of classical flight simulators. The SIMs were originally based on the M1 driver trainer, and there was no comparable legacy system in terms of capability. For SAF, vehicle/crew mobility is approximated by the crew behavior model, which inputs appropriate behaviors to allow the vehicle to perform in an automated manner.

For both SIMs and SAF, WES-provided mobility data permits a range of different types of paved surface, soil type, season and vegetation, and both dry and wet conditions. SIMs combine terrain coefficients with other inputs, while SAF uses tables with 30 specified terrain types. With SAF propulsion force is calculated using the vehicles velocity curves during acceleration. These velocity profiles are generated by the NATO Reference Mobility Model (NRMM), and are a function of vehicle type, terrain type, terrain slope, throttle position, and gear position.

SIMs account for terrain slope using weight vector resolution in 3 dimensional space. Model processing combines weight components and other simulation aspects in horizontal degrees of freedom to get the effects of slope, without use of traction or other terrain interactions. The SIM includes the M2/M3 slope

indicator which provides a check that the vehicle is on a sufficiently flat surface for TOW firing. SAF uses WES mobility data tables with 6 gradients of slope. Velocity for SIMs takes into account the vehicle engineering data, terrain coefficients, and operator inputs. In addition, dynamics such as power loss during steering is taken into account. For SAF, velocity is based on WES data curves/profiles.

The turn rate for SIMs is based on terrain coefficients, transmission engineering data, combined with the vehicle simulation and operator inputs. It is sensitive to terrain, velocity, load, power losses, and operator control. By integrating these factors, damage such as thrown track, rollover, or collisions can realistically occur, normally as a result of operator error. Thrown tracks are determined by track side-forces resulting from the interactions between terrain, soil type, velocity, operator error, and similar conditions. Similar input determines roll-over and vehicle drowning. Collision reactions are based on a Newtonian momentum model providing full reactions in longitudinal, lateral, and yaw directions, and can account for multiple collisions and angular momentum effects.

Turn rate and collisions are simplified in SAF. The behavioral inputs or the crew behavior model precludes errors or accidents from occurring by automatically slowing the vehicle to a safe but high speed (around turns) or by avoiding obstacles (steep slopes, non-fordable water, no-go terrain, terrain features with collision volumes, etc.). When a SAF vehicle collides with any of the above obstacles, the vehicle will come to a stop.

For towing, SIMs add additional resistance in the longitudinal direction based on towed vehicle mass and towing vehicle longitudinal velocity. All normal model processing is still fully active. For towed SIMs, Newtonian spring-damper forces are added based on towing vehicle position, and its effect is calculated. For dust trails, 1 of 3 sizes/types (or none) is generated using synthesized factors including vehicle velocity.

For SAF, thrown tracks are not a function of mobility, but may occur as discussed in damage assessment. Roll-over and drowning are also not used as part of mobility. In general, SAF avoids No-Go locations such as steep terrain and drowning-depth water. SAF vehicles are placed on the terrain with three contact points (two in front and one in the center rear). SAF collision and towing use modified versions of SIMs algorithms.

Findings and Concerns - Factors impacting mobility are reflected in the components of the system or its behaviors. Differences between SAF and SIMs are that SIMs have interactive crew/operator input, while SAF uses software to model operator decision logic. Also, SAF has no discrete physical model for vehicle components, but uses empirical output data and algorithms to describe how the vehicles move. SAF interaction with the terrain is an approximation of the many factors that impact mobility.

The main terrain difference between SAF and SIMs is that the visual and the SAF correlated database are designed for different purposes. SAF uses the SAF correlated database to determine areas of no-go or restricted terrain types. The SIM must rely on visual cues such as terrain skin texture and cut & fill slope and make their own determinations on mobility suitability. In several experiments the SAF halted or avoided areas of no-go terrain while the SIMs occasionally had problems cueing on the visual scene causing the vehicle to flip or get stuck. Terrain types dry peat (no-go) and river fordability determination have been identified as major issues in this area for the SIMs being able to cue on the terrain. Much of this problem comes from the design decision, to make the skin texture and features in the visual database match with the vegetation defined in the raw source data instead of the underlying soil type. Often the same vegetation type will have several different types of underlying soil. The visual systems design also provides different texture patterns to different areas of the same terrain to avoid the synthetic appearance of having a common texture for every terrain type. Being able to cue off the soil type is turning into a big issue for the SIMs.

Most vehicle components do not cause differences between SAF and SIMs. Engine, drivetrain, and fuel availability are important to mobility, but no significant differences are expected in their representation. The track and suspension interacts with the terrain and has the potential to create discernible differences between SAF and SIMs. SIMs account for most suspension system components, and outputs vehicle bounce, pitch, and roll, whereas SAF will not show these actions. SIMs also display the results of artificial undulation (rocking) in the pitch direction even over unvarying terrain. SIMs track and suspension effects will be visible (and will be even more drastic for braking).

The turning rate and radius is derived from AMSAA data. However, SIMs respond realistically to operator input including mistakes, and models problems such as fantailing, sliding out of a turn, or even overturning.

SAF does not model these actions; when a SAF vehicle reaches maximum turn conditions, the crew behavior model either causes the vehicle to slow, or use a wider turn radius (but still conduct a controlled turn). SIMs also take into account the large horsepower losses experience during turns, due to sideways dragging of the tracks. SAF does not account for this (it has no requirement to simulate engine functions), and is able to conduct turns at higher speeds without losing control. It is possible that during movement on winding roads, SIMs may have difficulty keeping up with like SAF tracked vehicles.

Along similar lines, the velocity/acceleration of SAF may differ from SIMs, in that the crew behavior model may cause SAF to move at an acceptable, but rapid pace (as defined in WES-supplied vehicle velocity curves), while the SIMs may move slower due to human operator uncertainties in terms of cues, command and control, or other factors. SIMs driving cues include visual effects, vehicle controllability, seat sub-woofers, artificial undulation, and terrain-induced bouncing, pitching, and rolling. These cues are likely to result in lower speeds. Similar factors pertain to braking/deceleration, again resulting in differences between SAF and SIMs. Suspension reactions exhibit visual differences as SIMs move using several degrees of freedom including pitching while moving over undulating terrain or during braking or turning operations.

Other effects such as towing, collisions, damage, and recoil also display differences between SAF and SIMs. A full range of reactions are provided for SIMs towing and collisions. Collisions can have effects in the longitudinal, lateral, and yaw directions, including glancing, rebounding, rotation, and slowing reactions, and can respond to multiple simultaneous collisions. SAF reactions will be simplified, producing reactions only in the longitudinal direction. For mobility-related damage such as thrown track or roll-over, SIMs synthesize terrain coefficients and vehicle velocity to determine the actual reaction under various conditions. Damage such as drowning can also result if operator error results in placing all road wheels in drowning water depth. SAF has much lower level of detail with regard to these actions. The SAF control mechanism in vehicle simulation (see behaviors section below for more details on the SAF architecture) is ideally operated at a 15 hertz rate to synchronize with the visual systems refresh rate. However the framework mechanism within the design degrades this rate when the processor becomes loaded and unable to maintain the 15 hertz rate. Typically the large numbers of vehicles (upwards of 50) and varying computational

loads, causes SAF to frequently operate in a degraded mode. As the frequency is reduced the ability of the entities to maintain stable vehicle control is reduced. In areas such as forests with numerous high density collision volumes, and steep turns the SAF entities may collide with the obstacles before the entities direction can be adjusted. This may cause the SAF designers to review the possibility of ignoring collisions under certain circumstances.

3.3 Target Acquisition

Of all correlation issues, target acquisition is widely viewed as the most significant, based on experiences with legacy systems such as SIMNET and CCTT user exercises. Typically the acquisition abilities of the SAF and SIMs are greatly mismatched. In SIMNET exercises, the SAF operators would typically limit the opening ranges of the SAF to avoid the SAF destroying the SIMs before they could detect the SAF.

The target acquisition ability supported by the SIMs is primarily dictated by the fidelity level provided by their Computer Image Generation (CIG) and display system. This fidelity is established somewhat by the state of the art in visual technology but more so by cost considerations. The contrast and resolution of the visual systems used in these moderately priced SIMs fall noticeably short of real world capabilities. The CIG is operated at a 15HZ update rate as a design compromise which enables the rendition of higher density imagery. The performance penalties of 15HZ, e.g., image stepping and multiple imaging, are experienced and degrade target acquisition performance. Additionally the CIGs overload management system sometimes compounds this degradation by modifying or eliminating scene elements critical to acquisition performance. This reduces resolution of the image while a sensor is scanning and forces SIMs into a reduced scan rate. The commander's popped hatch (CPH) visual system uses an area of interest scheme whereby only about forty percent of the field of view is presented at the CCTTs highest resolution. The rest of the CPH area has degraded resolution, creating some limits for the commander's view and scanning abilities. Finally, many of the SIMs vision assets have fields of view which are smaller than the design basis vehicle.

The SAF simulation uses the Night Vision Electronic Sensors Directorate (NVESD) target acquisition methodology (herein referred to as ACQUIRE methodology) to represent target acquisition sensor performance for Direct View Optics (DVO), Image

Intensifiers (I2), and thermal InfraRed (IR) systems. ACQUIRE methodology is currently used in constructive Army combat simulation models.

The ACQUIRE methodology, adopted by the Army in 1993, utilizes the same equations as the obsolete Night Vision Laboratory (NVL) methodology but requires modified input data. The ACQUIRE methodology differs from the NVL methodology in its use of a modified (line pair) criteria for the various level of target acquisition, in its use of a two dimensional representation of target size, and in its use of a two dimensional Minimum Resolvable Contrast (MRC) or Minimum Resolvable Temperature (MRT) curve. This methodology and associated assumptions are presented in the following sections.

Basic Definitions, Assumptions and Limitations: The ability to acquire a target in a particular environment is a complex function of not only the observer's visual perception of an image, but the object's size, shape, color and the background's scene luminance and thermal characteristics. Factors such as clutter, motion, camouflage, obscurants, etc., either enhance or degrade an observer's ability to detect, classify, recognize and identify objects. Training is a basic requirement to enhance an individual's target acquisition capability.

Definitions: Associated with ACQUIRE methodology are distinct factors that affect how, when, and where a target will be detected and ultimately upon by SAF. The AMSAA definitions of these factors are shown below. It was shown during BCR 174 that minor differences in terminology between the US Army Gunnery Manual definitions of similar terms and the ACQUIRE definitions do exist. These have been mapped to the ACQUIRE definitions below;

Field of Regard: The field of regard is the angular portion (horizontal and vertical) of the surrounding environment over which a sensor is moved to search for targets.

Field of View: The field of view is the angular portion (horizontal and vertical) of the surrounding environment visible through a sensor at any given instant of time.

Target Acquisition Levels: The following military definitions that comprise target acquisition levels are defined as follows:

Detection: The ability to distinguish an object of military interest in the field of view (FOV).

Classification: The ability to distinguish a target by class, e.g., tracked vehicle, a helicopter, or a wheeled vehicle.

Recognition: The ability to distinguish between different categories of targets within a class, i.e., tanks versus armored personnel carriers (APCs) in the tracked vehicle class.

Identification: The ability to distinguish between specific models of targets, i.e., a T72 tank versus a M1 tank.

Acquisition Criteria: The Johnson Criteria is the methodology used for the basis of target acquisition in the ACQUIRE methodology. The criteria equates the number of cycles per milliradian (cy/mr) or line pairs (LP), using a standard target board, with an acquisition task in which half of all observers can resolve the target. The acquisition levels and criteria are provided in Table 1 below.

Table 1. Target Acquisition Line Pair Criteria.

Acquisition	CRITERIA (cy/mr)
Detection Level	0.75
Classification	1.5
Recognition	3.0
Identification	6.0

Assumptions and Limitations of the ACQUIRE Methodology. Verification, Validation and Accreditation tests conducted on the ACQUIRE methodology indicate that, for FOV only search, the model can accurately predict, to within 20 percent, the range at which given values of target acquisition probability are achieved. These results are restricted to those conditions that can be accurately represented by the methodology. There are a number conditions of interest that cannot be represented or accurately modeled by the ACQUIRE methodology. These limitations are discussed below as follows; Modeling of Moving Targets, The ACQUIRE methodology is not designed to model the effects of target motion on target acquisition. A moving target's motion acts as a visual cue in the target detection process; therefore, the probability of detecting a moving target can be significantly greater than the probability of detecting an otherwise identical stationary target under the same set of conditions. AMSAA uses a line pair criteria of 0.5 for modeling detection of targets with a significant radial velocity component across the line-of-sight. This line pair criteria was chosen because earlier research suggested that a line pair criteria of 0.5 could be used to model detection for a zero clutter condition. In addition ground clutter, pinpoint effect such as muzzle flash or dust trails, and multi-target acquisition and not modeled by the ACQUIRE methodology.

TARGET ACQUISITION

(PRIMARY COMPUTATIONS)

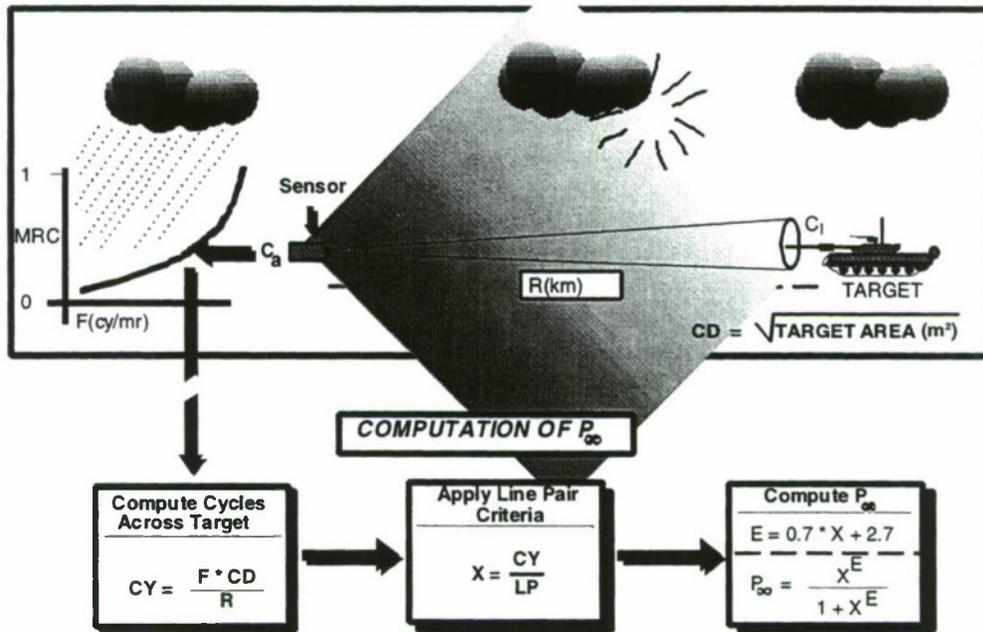


Figure 1. Target Acquisition Computation Overview

Figure 1 provides an overview of the primary inputs and computations employed by the ACQUIRE model for a visual sensor. The top part of the main factors which are target, sensor type and intervening atmosphere. The target is described in the model by input values such as inherent contrast (C_i) and characteristic dimension (CD). The sensor is described in terms of a sensor resolution curve, MRT. The MRC/MRT curve is unique to each sensor and is a function of spatial frequency. The atmosphere is depicted in terms of atmospheric attenuation, visibility range, ambient light level and sky to ground ration. The lower portion of the figure provides the computational steps used to compute the acquisition probability given an infinite amount of time. This quantity is known as "P infinity" and is used to compute the average acquisition time and acquisition probability given a finite time P(t).

Findings and Concerns - For the search process and the rate at which targets are detected, recognized, and identified, SIMs are dependent on crew skill. For the search process, SAF uses AMSAA data and a scan rate. For the rate to detect, recognize, and identify targets, SAF uses AMSAA's ACQUIRE algorithm. Basically,

the difference is that SIMs use a crew and SAF uses AMSAA and other data.

Currently the SAF disregards targets that are less than 30% exposed. This makes detecting targets that are in defilade or hull down positions difficult to acquire and may provide a unfair advantage to the SIMs. The CCTT design treats constructed defilade positions as a relocateable object on the terrain skin. These object tend to stand out from the color and texture of the surrounding skin making them more noticeable to the SIMs. Thus the defilade positions or other relocateable harboring SAF entities may be unfairly acquired by the SIMs.

Visual representation of terrain has an impact on target acquisition for SIMs and SAF. At ranges beyond 2,400 meters, terrain objects can fade, particularly at more distant range, such as 3,500 to 4,000 meters. This fading is due to the image generator's "load management" of the number of polygons used to represent terrain objects while maintaining an appropriate level of visual fidelity. For example, if there is a building in front of a potential target, the image generator can fade the building due to "load management". With the building faded, SIMs now

have line of sight to the target behind the object that was faded. However, SAF will continue to see the building and would not have line of sight to any potential targets.

There may be a difference in the rate in which SAF and SIMs detect, recognize, and identify targets. Using the sensors in SIMs, detection, recognition and identification of targets is dependent upon crew skill. SAF knows the location and identity of all targets in a sensor's field of view and may detect movement quicker than SIMs because it uses the ACQUIRE model while SIMs depend on its crew. Currently, SAF will not detect muzzle flash, smoke, or dust clouds. Given the data (AMSAA does not have muzzle flash data), physical models can develop the code for these signatures; behaviors cannot use these signatures to detect line of sight to actual targets because muzzle flash, smoke, and dust are not considered targets by SAF. The search process itself is discussed under behaviors below.

Target contrast is a major issue with the ACQUIRE model. The SAF performs contrast in a relatively simplistic manner by determining the terrain type (e.g., forest) the target is located on. A adjustment factor is provided for that type which is used in the model. The SAF does not have processing time to search the background behind the target to do a detailed analysis of the target contrast. For example a tank has the same contrast factors being well positioned in a tree line as opposed to being in the open just outside the tree line assuming it is located in the same terrain area. As of the writing of this paper, the SIMs were just finishing tuning the thermal sensors. The correlation of thermal abilities will likely raise a number of issues.

3.4 Behavioral Aspects

The Behaviors Computer System Component (CSC) provides the basic decision logic and control for the SAF entities. Table 2 shows the structured layers of dependency within the CGF design. The components higher in the Table 2 list can "with" or directly call components lower in the list. For components lower in the list to send information to components higher in the list they must use a callback, which is similar to leaving a message in a data file to get back to you. This structure provides for a well organized dependencies structure and avoids complications such as cyclical dependencies. Within the CGF design, behaviors includes sub-elements such as; Crew Behaviors, Small Unit Tactics, large unit tactics, and order decomposition. Vehicle simulation includes; hulls, sensor, turret, weapon, resource, special effects

and damage assessment functionality. Environment includes; line of sight determination, munitions impact detection, route generation and verification, obstacle avoidance, relocatable and preposition object management, collision detection, height of terrain/surface type, cover and concealed position location and environmental effects.

Table 2. Layered Structure of the CGF CSC

Simulation Manager CSC
Behaviors CSC
Vehicle Simulation CSC
Framework CSC
Environment CSC

A typical target engagement scenario has the following flow of events;. Sensors determines which targets are in the sensors field of regard (scan area) and passes them off to the environment. The environment line of site function uses a 20 point raster to determine the amount of the target which is visible by performing a intersection search between the 20 points on the target and the sensor. If the target is blocked by a partial transparent object such as a tree, the visibility is adjusted according to the objects opacity factor. A percent visibility is returned to sensors, which uses the ACQUIRE model to add in factors such as time of day, weather effects ,target size, and sectors the sensor is scanning, to determine to what level the target is visible (e.g. identified, detected ,classified). Sensors builds a spot list and passes it back to crew behaviors, which generates a spot report (for new sightings). The crew behaviors may move the gunners sensor over to a detected target to magnify it and try to get a identification. If a number of targets are identified, unit behaviors will identify the "most dangerous target" for the vehicles weapon(s) to engage. This criteria is defined by current doctrine which looks at factors like; does the target have the ability to destroy the vehicle, how close is the target to the vehicle, and is the targets weapon systems (articulated parts) pointed at the vehicle. This provides a simplified example of one of many complex interactions between behaviors and the CGF primitive functions that occur. The behaviors provide a number of correlation issues

Findings and Concerns - Behaviors define the rules of engagement for the SAF entities. Current doctrine specifies that targets will not be engaged unless identified and is the primary rule of engagement represented in SAF. This policy and the knowledge of where all targets and entities are on the battlefield

prohibit fratricide by SAF. However, a SIM, depending on situational awareness received by the human crew, may fire on a lesser visible target (e.g. not identified) which may cause fratricide to occur. Although initial setup of CCTT attempts to match rules of engagement between SAF and SIMs, difference due to human interaction should be expected to occur. In the sense of a "fair fight", the SIMs would have an unfair advantage if the rules of engagement differ between SAF and SIMs.

In the current experiments the SIMs typically searched , based on situational awareness, and concentrated on known hiding areas such as tree lines. SAF gunner and tank commander search sectors are predefined by doctrine and may miss less obvious targets. To eliminate some of these differences between SAF and SIMs, the current target tracking methodology is being modified to account for target persistence. If a detected target were to move into a hiding position, the SAF process would drop it from the target list and the vehicle would resume searching for targets. This may provide a major advantage to the SIMs which could briefly hide, until the SAF vehicle started to scan again and then pop out and engage the SAF. Currently, fixes which have SAF concentrate on the last known location of a object for a limited duration of time are being implemented., thereby recognizing target persistence in a manner similar to that represented in some constructive models and simulations.

3.5 Rate of Fire

The Rate of fire methodology for SAF is based on a series of variables for which data has been provided to SAF. The variables are: the weapons slew rate to point the weapon at the target; the weapons load time which is static for automatic loading vehicles such as a T-80 and stochastic for breach loaded vehicles as a M1A1; the lay time for the first round which is stochastic based on a table which provides different median time values based on range and weapon system; the time of flight for each round to the target ; the subsequent round lay time which is stochastic based on a table that provides the median time based on the weapon system. Adjustment factors are suggested for target motion and firing while moving. The data provided is based on average combat conditions, average crew proficiency, stationary firing weapon and a stationary target. CCTT requires that the SAF operator be able to adjust crew proficiency. This is performed by adjusting the rate of fire factors and the delivery accuracy to reflect the crew competency. The combination of data allows SAF to

generate simulated events which may be used as checkpoints for crew proficiency. Again, it must be noted that SIMs performance is greatly affected by the human interactions within the vehicle platform.

Findings and Concerns - Some question exists as to what factors should be adjusted to reflect crew competency. The rate of fire methodology has created some concern based on the fact that during some tests , crew members felt that the SAF fired at a unrealistic rate. As with all data provided, the various combinations involve some risk of providing credible results when implemented. In tests conducted with master gunners acquiring, detecting and identifying targets, ModSAF results were very comparable, indicating that the data sets used may be representative of average or expert crews. While this area is not viewed with the most urgency, future correlation experiments involving master gunners may be planned to check rates of fire combinations.

3.6 Delivery Accuracy

SAF computes the delivery accuracy based on firing at the center of a target (direct fire) or at a location (indirect fire). The direct fire methodology is based on having the weapons system aim at the center of mass of the target. The intended aimpoint is adjusted based on data tables which provide variable and random bias adjustment factors based on weapon system and range. Additional variance is added for either moving targets or moving while firing. If the target is moving, the new adjusted aimpoint is projected in front of the target based on it's velocity and heading. The "fire at location" is adjusted based on the variances alone. Typically the variances for indirect fire weapons and much larger than direct fire weapons. In addition the SAF Weapons code multiplies the adjusted bias based on the crew competency, to decrease variance for expert crews and increase variance for novice crews..

Findings and Concerns - Since SAF fires at the center of mass instead of the center of the exposed target area, concern exists now to handle unique situations such as targets in defilade positions. The method that SAF uses to project munitions flight based on target velocity projections has shown that in situations where the target is moving on irregular terrain (up and down Z values) it may increase the odds of missing the target. During several exercises SAF seemed to have exceptional accuracy compared to the SIMs on fully exposed targets. This area will likely be reviewed in future BCR 174 testing.

4. Test Results

The first BCR 174 test scenario was designed to be a daytime run with unrestricted visibility. Both SAF and SIMs (M1A1 & M1A2) used only Direct Vision Optics (DVO), no thermal capabilities. A combination of T80, BMPs and DI targets where all emplaced in tree lines, out in the open and on the sky line. Several targets where emplaced so they only had a limited window of opportunity from a fixed direction to be detected. All targets where stationary. The detecting vehicle went down a 8 Kilometer course at a search speed of 25 kms per hour. All targets where emplaced to be acquired between 500 and 4000 meters from the course. A total of 3 targets sets where developed of 4 targets each for the course. The crews ran all 3 target sets as one run, a total of 10 runs where performed for both SAF and SIMs(with different crews).

Findings and Concerns- The SAF proved to be slightly to significantly better in target acquisitions then SIMs in most circumstances. SAF was significantly better in acquiring DI targets even at extended ranges. The SIMs had great difficulty acquiring DI targets in tree lines because of their small size and the ease they where mistaken for tree

trunks. The feeling among the SIM operators was that DI were thermal targets. In future test runs when the SIMs use thermal sensors the DI should be easier to acquire. To fix this problem from a SAF perspective, the contrast factors will be lowered in the ACQUIRE model for DI targets to make them less detectable.

SAF proved to be able to detect all targets even if they only had a limited window of opportunity, or required rear or side vision to detect. This likely comes from the fact that the M1A1 is modeled with a driver with 120 degrees of forward unity vision, a loader with 360 degrees of unity vision ,a commander with 360 degrees of unity vision and a gunner with a magnified site. The 3 unity sensors increased the odds of getting a detection on the target, regardless of its direction with respect to the searching vehicle. Once a detection was made the SAF vehicle would bring its gunner's magnification sensor on the target and identify it. In the SIMs the loader and driver sites proved to be no where near as effective in detecting targets. To fix this problem we are having AMSAA look into the effectiveness of commanders and loaders in acquiring targets. Future test will look into the effectiveness of each searcher in acquiring targets for both SAF and SIMs.

Target Number		MODULE ACQUIRED DISTANCES (Meters)					SAF ACQUIRED DISTANCES (Meters)				
		1	2	3	4	5	1	2	3	4	5
1	BMP	1587	1537	2055	1846	1415	1943	1944	1943	1943	1939
2	DI	NA	NA	NA	NA	NA	1561	1821	1829	1862	1820
3	BMP PLT	1917	NA	1922	NA	1757	1922	1905	1910	1924	1923
4	T-80	3706	1586	2636	1923	1711	3704	3862	3989	3974	3960
5	BMP	2994	2217	2343	2250	2278	2638	2389	2974	2937	2767
6	T-80	NA	604	566	NA	NA	612	613	613	613	610
7	DI	195	840	844	290	1161	2264	917	2098	2177	2007
8	BMP	1666	1702	1750	NA	1757	1736	1730	3436	1732	1731
9	T-80	1214	1194	1369	1221	1306	1404	1396	1374	1402	1405
10	T-80	763	1400	1013	NA	1016	3976	3992	3699	3984	3993
11	T-80	NA	NA	2375	1943	NA	3141	3154	3317	3303	3308
12	T-80	3796	3097	3720	3320	2057	3938	2590	3844	3646	3595

Table 3 - Provides a sampling of data from 5 of the 10 SAF & SIM runs. Acquired distances are when the vehicle running the course engages the target. Which is an identified target in the ACQUIRE model for SAF. NA- Not Acquired

5. Conclusions

The specification for SAF contains the statement, "The SAF software shall provide the SAF units behavior based on the operator's inputs from the workstations and will be indistinguishable from the manned simulators". Meeting this requirement via the CCTT BCR 174 effort has many challenges. Without correlation, CCTT will be difficult to accredit as a training system, in the ongoing VV&A process. The correlation effort also offers challenges to data

providers such as AMSAA to develop data sets that are modifiable based on the limitation and capabilities of the systems synthetic environment. One of the by products of the BCR 174 effort will be test cases that could be used in the future to gather data for analysis. The immediate test plans include; test cases with BLUFOR targets mixed among OPFOR that look into the level of detection and rules of engagement, moving targets, different target types, thermal detection , weather, obscurants and time of day.

6. References

AMSAA (1995) The 1995 Compendium Of Close
Combat Tactical Trainer Algorithms, Data,
Data Structures and Generic Systems
Mappings

IDT CGF Physical Model Team (Feb. 22,1996)
CCTT CGF Dynamic Behavior Design Synthesis
Report

7. Authors' Biographies

Edward V. Chandler is the on-site Subject Matter Expert (SME) for CCTT SAF. He has developed, written/reviewed extensive natural language CISs and is involved with systems testing for SAF. He has 30 years experience in the Army's joint tactical operation planning and execution, and has commanded from platoon through Brigade.

Henry Marshall is the government lead on the CCTT SAF integration team. Most of his duties have been in software and CGF acquisition support for STRICOM and NTSC. He received a BSE in Electrical Engineering and an MS in Systems Simulation from the University of Central Florida.

Brian R. McEnany, a 1962 graduate of the US Military Academy, received his MS in Operations Research and Statistics and an MS in Management Science from the Rensselaer Polytechnic Institute in 1970. He led the knowledge acquisition and combat instruction sets effort as lead subject matter expert for the development of OPFOR and BLUFOR combat behaviors in CCTT.

John G. Thomas is an Operations Research Analyst in the simulations Branch, Combat Integration Division AMSAA. Mr. Thomas is AMSAA's lead analyst for the CCTT and ModSAF VV&A efforts. Mr. Thomas has a Master of Arts degree in Mathematics.



Validation of Individual Combatant Simulation Using a Model-Test-Model Approach

George R. Mastroianni
Natick Research, Development, and Engineering Center
Natick, MA 01760

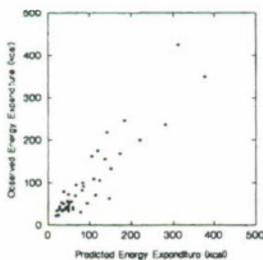
Victor E. Middleton
Simulation Technologies, Inc.
Dayton, OH 45402

1. Abstract

We conducted a field study to validate the energy expenditure prediction algorithms of cross-country dismounted movement in our small-unit simulation system (the Integrated Unit Simulation System - IUSS). March rate, terrain grade, and individual energy expenditure were estimated for a lengthy route (approximately 70 miles) over mountainous terrain. While energy expenditure predictions from the model appear to be valid, the accuracy of these predictions is heavily dependent on knowledge of march rate. We explored several methods for generating valid march rate predictions. A fuzzy-logic approach appears promising especially in that it may have wider application in the representation of human behavior.

2. Introduction

Figure 1



The IUSS is a comprehensive computer simulation environment emphasizing dismounted infantry operations. It employs a sophisticated suite of physiological, ballistic and chemical models to relate soldier capabilities to mission demands, environmental conditions, and combat outcome. The IUSS contains physiological models that relate work intensity and energy expenditure to the combined effects of load, terrain conditions, environmental conditions, soldier march rate, and clothing. The models consider the heat transfer properties of the

clothing and equipment associated with each simulated individual and compute predicted heart rate, skin temperature, core temperature, and other indices of soldier physiological state at frequent intervals. Figure 1 shows that observed energy expenditure values bear a close relationship to field observations, suggesting that this component of our system is valid.

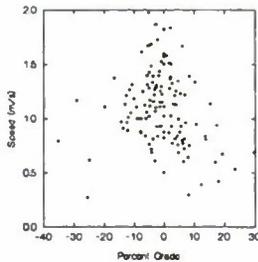
An important long-term goal of ours is to improve our understanding of the relationship between terrain characteristics and voluntary march parameters such as speed. Knowledge of soldier speed and other mobility parameters is crucial to realistic simulation of dismounted movement. Overestimating speed in our simulation leads to unrealistically high energy expenditure and body temperature predictions; underestimating speed has opposite effects.

At present, prediction of soldier speed relies heavily on the terrain grade; on steep uphill, we expect soldiers to slow down; on downhill grades, we expect them to walk faster. There is no generally accepted method for predicting march rate across complex terrain. We used our field observations to develop a first approximation to a method of generating these predictions, at least in the context of cross-country movement without anticipated enemy contact. We have not yet addressed the problem of dismounted movement under more tactically demanding conditions, which presents a much more complex problem.

3. Initial Findings

We expected to see a strong relationship between terrain grade and march rate in our field observations

Figure 2

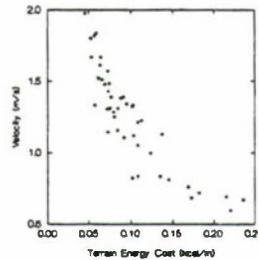


As Figure 2 illustrates, however, there is considerable variability in march rate observed for segments with very similar grade.

In laboratory research, grade is generally fixed on a treadmill, and other factors potentially contributing to workload held constant. In the field, though, soldier choice of grade is heavily influenced by factors other than grade. Examples of these are substrate type, soil condition, and terrain smoothness, or the consistency of grade. This last factor is important because our method of grade measurement in the field provides only an average grade over the segment distance, and is insensitive to frequent shifts or reversals of grade that characterize some terrain. Under these conditions, our measure of grade does not faithfully reflect the workload experienced by a soldier. A further complication in proceeding from laboratory or field data to simulation is that terrain data files available to us do not support higher resolution analysis of grade.

To overcome some of the shortcomings of our grade measurement method, we defined another measure of workload: energy expended per meter. We estimated the energy expended over a segment using the average heart rate for that segment, and divided by the distance. This measure confounds grade and speed, because workload is partially determined by the speed chosen, but it implicitly includes the contributions of other (as yet unspecified) factors. Our data, as shown in Figure 3, illustrate that energy expended per meter (kcal/m) is more systematically related to speed than is grade.

Figure 3



Because the definition of speed is so important to achieving good simulation results, we attempted to explore the use of energy expenditure measures from our field study to test some alternative methods of predicting soldier speed. We tried (1) predicting speed based solely on grade and (2) predicting speed by assuming that soldiers tend to maintain a relatively constant work intensity. We tested these approaches and compared the accuracy of resulting speed predictions. Predicting speed directly from terrain grade was accomplished using the equations of the regression lines relating grade and speed. (Separate regressions were used for uphill and downhill segments). This method produced a correlation of .55 between predicted and observed speeds; grade accounted for 30.25% of the variance in speed.

After inspecting the speed prediction residuals plotted against energy cost per meter, we selected segments with relatively extreme values. We adjusted the predicted speed of segments with very low terrain-related energy cost ($z < -.5$) upward by .85 m/s, and the speed of segments with very high terrain-related energy cost ($z > .5$) down by .25 m/s. This arbitrary adjustment increased the correlation between predicted and observed speeds to .81, and accounted for 65.6% of the variance in speed.

To predict speeds based on an assumed constant work intensity, we computed speeds for our segments that would produce a work intensity of 466 watts (the overall mean). Energy expenditure prediction for downhill locomotion is not as well understood as is prediction on level or uphill terrain. The equations that form the core of our physiological model do not apply to downhill grades; speeds computed in this way for downhill segments are unrealistically high. The correlation between these speeds and observed speeds is -.11. We adjusted these predictions by substituting speeds computed for corresponding uphill grades on downhill segments. These values work better than those computed using negative

grades, correlating .51 with the observed speeds. We adjusted these predicted speeds for energy cost per meter using the same method described above for the grade-based speed predictions. The adjusted predictions correlated .77 with observed values, accounting for 59.3% of the variance in speed. Both methods show promise as methods for generating useful speed predictions. While the first method requires specific knowledge of the grade-speed regression for these data, and the second requires an estimate of the overall work intensity sought by soldiers, both methods produce much more realistic speed predictions than the practice of using a constant speed for all segments.

4. A Fuzzy Logic Analysis

While some of the variability in speed is undoubtedly related to measurable terrain factors, we suspect that speed regulation, like many other aspects of human behavior, will never yield completely to deterministic schemes. Humans are organisms of such staggering complexity that even apparently simple functions such as speed regulation may be influenced by a host of variables and considerations, such as past experience, knowledge of future demands, cumulative fatigue effects, expectations of imminent rest, and the like.

It is unlikely that quantitative data will ever be available for more than a small fraction of the human behaviors we need to represent in combat simulation systems. This assessment prompts us to look for a method that permits us to use both the empirical data available and the insights provided by subject matter experts to produce valid representations of complex human response as an improvement over strictly data-based or insight-based approaches. For this effort, we explored the use of fuzzy logic concepts to provide a method of integrating insight and data, and in particular to develop a model of speed/terrain relationships.

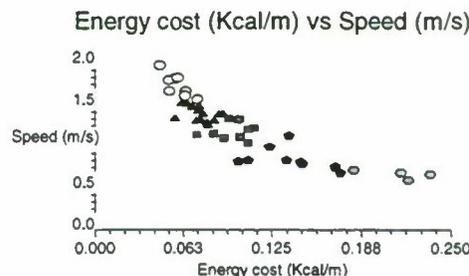
We concentrated on energy expended per meter as the primary determinant of speed. Clearly, additional research is needed to permit us to decompose this global measure of terrain difficulty into constituent factors, including grade, frequency and magnitude of grade changes, substrate type and condition, and the like. Our use of this measure is predicated on the assumption that at some future time, we will be able to account for the same variance in speed that is now related to energy expended per meter using a larger set of factors. The analysis presented here is meant to

explore the potential utility of the methods, not to accomplish a definitive treatment of the problem.

4.1 Data Clustering

We included speed and energy cost per meter as input variables to a popular commercially available fuzzy logic software system. Five clusters of points were identified. Adjective sets were chosen as descriptors of these clusters for both energy

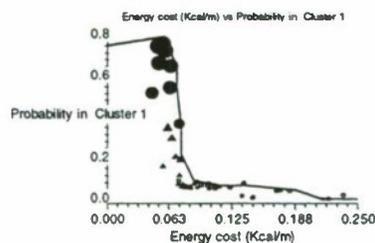
Figure 4



cost and speed. Energy cost was assigned five descriptors: Very Low, Low, Moderate, High, Very High. Speed was assigned Very Slow, Slow, March, Fast, Very Fast. A simple set of five rules was defined, relating energy cost to speed. (Higher energy costs produce slower speeds). Figure 4 shows the data as clustered.

Membership curves were constructed for each variable for the five clusters. These curves were produced by plotting the points against the probability of membership in each cluster; a function was then "eyeballed" in to

Figure 5

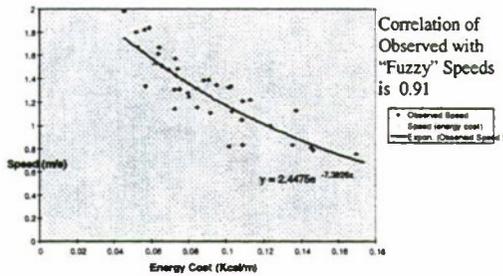


establish a continuous probability of membership function over the range of energy cost and speed observed. An example of the construction of one of these membership curves is shown in Figure 5.

4.2 Fuzzy Prediction

These membership curves and the five simple inference rules were used to produce a set of predicted speeds based on the observed energy cost values. (The “centroid” method was used to defuzzify the speed predictions.) As expected,

Figure 6



the predicted “fuzzy” speeds correlated quite well with the actual speeds - the Pearson correlation was approximately .91. Figure 6 illustrates these data.

Because this process essentially amounted to providing a curve fit to the available data, this result is unsurprising; however, it is encouraging to note that the prediction performance of this method is superior to other techniques that have been attempted. The constant work intensity method used earlier produces a correlation between predicted and actual speeds of approximately .77. The fuzzy method, then, does as good or better a job at predicting speed from energy cost than other methods we have tried.

4.3 Current Limitations and Potential

The fuzzy analysis we performed was limited by our data. As discussed above, we chose to work with the single variable most related to speed, energy cost, and our results amounted to a curve fit expressing speed as a function of energy cost. In this particular instance, as also illustrated in Figure 5, the data can be described about as well using a simple exponential curve fit. The value of this analysis to us, however, is not the demonstration of a new curve fitting technique, especially given that existing techniques serve as well or better for the data at hand. Rather, we have explored the viability of a technique that we believe holds the best promise for extending our results into domains where current methodologies will not serve.

This belief (in the potential of rule-based inference schemes based on fuzzy logic) is driven by several factors. Human behavior is fundamentally fuzzy. Soldiers do not march across country at constant speeds, and they don't adjust speed in terms of fixed increments - they go a little faster, or a lot faster based on their perception of their environment and their psycho-physiological state. That state is another example of fuzzy: tired, afraid, hot, do not lend themselves to precise measurement.

Fuzzy inference rules are intuitive. They make sense to soldiers, and can be developed in conjunction with soldier inputs without the need for intense training or understanding of the formal algorithms involved. “Go slow if you are using a lot of energy to cross the terrain, go fast if it is easy” is more acceptable to the average individual than $y=2.444e^{-7.3826}$

Fuzzy inference rule sets can accommodate inconsistent or even contradictory rules. Human behaviors are frequently driven by complex and often contradictory impulses. Fuzzy inference engines resolve such conflicts by assessing the relative weights by which different rules fire, and by adjusting behaviors by corresponding degrees.

Fuzzy inference rule sets easily incorporate multi-modal distributions. In the example above, a simple plot of the bivariate data involved suggests an exponential fit, and these data support a regression fit to such an equation. In many cases, however, field data are not so suggestive, and often group about distribution modes or otherwise are more amenable to a piecewise fit to regression curves. The cluster analysis we used to derive fuzzy adjective membership sets is easily applied to such data, and in fact is actually more valid in such cases than it is for our current data

Fuzzy inference rule sets can accommodate new variables relatively easily. The complexity of curve fit techniques grows rapidly with the number of variables involved. Bivariate relationships result in two-dimensional curves, adding additional variables leads to multi-dimensional surfaces, and corresponding computational complexity. Fuzzy inference rule sets, on the other hand, integrate new information with the addition of new rules. If rules are based on variables about which soldiers have knowledge, sparse data can be supplemented with soldier-generated insights to produce realistic behaviors.

In the example presented here, the use of energy expenditure to predict speed is of limited use to our simulation because energy cost is not directly related to any of our current terrain base variables. If, however, we can establish that speed is well explained by a combination of grade, substrate type, and perhaps one or two other variables, we believe it will be much easier to encapsulate this relationship through modifications to our current rule set than to develop the complex surfaces defined by multi-variable regression relationships.

5. Summary

A key advantage of the fuzzy logic approach is that it facilitates the integration of insight and information. The development of rules and adjective sets is a kind of knowledge acquisition, in which information is combined with individual insight to produce a computational model that weights the relative contribution of various factors in determining an outcome. This process is ideally suited to the development of better simulations of human behavior, where empirical data is often sparse. The approach contains the possibility for continuous improvement, because rules can be deleted, added, or modified as new data become available.

6. Authors' Biographies

George R. Mastroianni is a research psychologist at Natick Research, Development and Engineering Center. He is interested in human performance measurement and the representation of human behavior in simulations.

Victor E. Middleton is a mathematician and consultant working for Simulation Technologies, Inc under a contract to Natick Research, Development and Engineering Center. Mr. Middleton is one of the principal architects of Natick's Integrated Unit Simulation System (IUSS). His recent interests include the development of advanced computational approaches for representing human behavior in simulations.



Using the Combat Instruction Set for Verification and Validation of Semi-Automated Force Behaviors: High and Low Intensity Case Studies

Damon D. Baker
Charles W. (Chad) Mullis
U.S. Army TRADOC Analysis Center - White Sands Missile Range
ATTN: ATRC-WE
WSMR, NM 88002-5502
baker@trac.wsmr.army.mil
mullis@trac.wsmr.army.mil

1. Abstract

Identifying a process both credible and efficient for evaluating semi-automated force (SAF) behaviors within the context of performing verification and validation (V&V) functions in computer generated forces (CGF) is not a trivial matter. This paper presents one approach employing the combat instruction set (CIS) in evaluating the behaviors of SAFs. A brief synopsis of the CIS precedes a presentation the general flow of the process necessary to first verify adequate behavioral functional coverage and then validate model performance against a real-world standard. This description of the method is punctuated with examples of its application in V&V for both the Army Modular SAF (ModSAF) and the Close Combat Tactical Trainer (CCTT) SAF. The behavioral V&V efforts for ModSAF and CCTT SAF are compared and contrasted to show similarity of the method in light of the great diversity in resource availability and allocation on the two evaluation efforts. Utility of the CIS in developmental efforts such as tracking software as it progresses to maturity is also explored.

2. Introduction

Evaluating semi-automated force (SAF) behaviors in modern simulations is not a trivial matter. The evaluation must first verify adequate behavioral functional coverage and then validate model performance against a real-world standard. There exist a nearly infinite set of combinations of conditions and terrain spaces in which the behaviors set may be assessed, and there are often severe limitations regarding both time and resources which are available to be applied to the verification and validation (V&V) process. Also, run-time complexity can mean that combinations of perfectly good code lead to

an unrealistic performance of the model at the applications layer. Thus, the task of designing an evaluation which is credible, yet remains capable of being completed on time and within budget, can be somewhat daunting.

One approach to performing V&V evaluations of SAFs which can be designed to meet even very limited resource availability entails application of the Combat Instruction Set (CIS) as the yardstick by which the behaviors are measured. In this paper we present a method which has been successful in using the CIS to conduct V&V of two emergent computer generated force (CGF) SAF simulations. But first you may ask, what exactly is a CIS, and why should I care about it?

2.1 Combat Instruction Sets

A combat instruction set is defined as a computer generated representation of a tactical combat behavior at a unit and platform level (McEnany and Marshall, 1994). While the units may include organizational levels of Battalion, Company, Platoon, Squad, Section, and Fire Team, the platform is defined as a representation of an air or ground vehicle, or a dismounted infantry, engineer, or scout entity at some organizational level. A platform object has modeling attributes which include a physical body (hull), the ability to mount a weapon system and sensor, carry supplies and ammunition, a cross-country maneuver capability, and the potential to house one or more crew members.

As an adjunct to the CCTT development effort, the U.S. Army Program Manager - Combined Arms Tactical Trainer (PM CATT) sponsored work to collate combat behavior source information into a single repository--the CATT

Task Data Base. Information on unit behaviors was gleaned from various Army regulations, field and technical manuals, Army Training and Evaluation Plans (ARTEP), Mission Training Plans (MTP), and Russian-Heavy doctrine source materials. This data set comprises in excess of 700 separate CIS, and is maintained, updated, and distributed under contract with the U.S. Army by Resource Consultants Inc., 3051 Technology Parkway, Orlando, FL.

The CIS describes the tasks, conditions and standards by which Army units and their subordinate platforms are required to perform various combat behaviors. Examples of these collective tasks for a BLUFOR tank platoon include **Execute a Wedge Formation (B0004)**, **Action Drill Left (B0009)**, **React to Indirect Fires Drill (B0013)**, and **Assault an Enemy Position (B0031)**. There are an additional 43 named CIS available for the BLUFOR tank platoon with similar CIS coverage for other unit types (infantry, scout, etc.) and sizes (company, battalion, etc.) in the **CATT Task Data Base**.

The process of including a behavior in the **CATT Task Data Base** includes receipt of a signature affirming validation of the behavior as described, thus obviating any need for verifying or validating the CIS itself. In short, the CIS is a natural language description of the doctrinally sound combat behavior set, and it is an excellent starting point for initiating the process of modeling and simulating the modern battlefield.

2.2 Verification and Validation

Verification and validation are required for all models, simulators, and simulations developed under the Army Model and Simulation Management Program (AMSMP). Guidance is provided by Army Regulation (AR) 5-11 and DA Pamphlet (DA Pam) 5-11. Verification is the process of determining that the model, simulation (MS), or simulator (S) accurately represents the developer's conceptual description and specifications. Validation is the process of determining the extent that the MS or S represents the real world entity.

In short, verification entails ensuring that a model is doing things right while validation comprises ensuring that it is doing right things. When applied to the SAF behaviors arena, a V&V effort can be as detailed (and costly) as examining individual lines of code, or as high level (and theoretically inexpensive) as saying "looks good to me." Most program managers

would prefer to pay for the latter while receiving the benefits of the former.

3. V&V Methodologies

The problem of matching an affordable level of effort with the regulatory requirements for V&V is faced by all CGF/SAF development programs. Analysts at TRAC-WSMR have been performing V&V evaluations of SAF behaviors which encompass both ends of the detail spectrum; at the very detailed low-end we have been assessing ModSAF; and at the limited detail high-end we have been evaluating CCTT SAF. However, the design process for both levels of intensity have been sourced from the CIS.

3.1 ModSAF Behaviors V&V

The very detailed low-end of the SAF V&V spectrum is represented by our work with the V&V of ModSAF. The purpose of the the effort was to verify, validate and accredit (VV&A) ModSAF as required in the Anti-Armor Advanced Technology Demonstration (A2 ATD) Technology Demonstration Plan (TDP). The purpose of the A2 ATD was to develop and demonstrate a verified, validated, and accredited DIS capability to support anti-armor weapon system virtual prototyping, concept formulation, requirements definition, effectiveness evaluation, and mission area analysis on a combined arms battlefield at the Battalion Task Force or Brigade level.

3.1.1 Verification Plan

The overall approach for verification of ModSAF entailed assessing the individual code libraries, and specific details including which libraries can be found in Denney (1994). In performing verification of the ModSAF libraries as well as the overall model the following subtasks were completed.

3.1.1.1 Documentation Review. Documentation for ModSAF 1.0 was supplied to TRAC-WSMR, and detailed reviews were conducted regarding: adequacy of documentation (determined by compliance to TRADOC 5-11 standards), clearness, completeness, and sufficiency to meet the intended purposes. As a part of the documentation review, the contractor acceptance test procedures were reviewed for compliance to the requirements document for ModSAF 1.0.

3.1.1.2. Algorithmic and Methodology Review.

Algorithmic and methodology review were done both by TRAC-WSMR and AMSAA. Standardized Army algorithms as delineated in AMSAA's compendium of algorithms were the primary standard for weapon systems performance and effectiveness.

3.1.1.3. Verification Software Testing (VST).

Stylized tests were conducted to insure: that the software would perform within the design limits and was adequate to meet performance needs; that, in the case of data voids, default values were used; and that the performance degraded in a graceful manner when operating near or outside of the designed limits.

3.1.1.4. Top Level Tests (TLT).

The goal of TLT was to assure that the libraries worked together as a unit. An integrity check of the overall model structure was necessary to assure that the model cycled correctly (i.e. all of the units were serviced for all applicable functions). The model was evaluated in a stand-alone mode as well as send and receive information mode which affects other simulations (i.e. simulators, AFORs, etc..) as well as its own simulation. The SAFsim is the primary tool to setup and run a ModSAF simulation while the SAFstation is the primary link to the DIS. The primary function of the SAFlogger is to record and playback simulation sessions. Each of these modules were tested separately, in as much as it was feasible, and they were also be tested as a modular unit.

3.1.1.5. Parametric Analysis. Wherever necessary, and as time permitted, individual libraries were tested, using parameter values which fell within the normal range of distributions for the selected variables under question. The aim of this exercise was to measure the affects on the simulation throughout a range of selected test values. While the library itself may function for all the selected test values, it may well be that some values for some variables would cause anomalies in the overall simulation or else cause unacceptable run times (times causing unrealistic responses).

3.1.1.6. Peer Review. A critical and detailed analysis of the model's internal representations and outputs by functional area experts was as much a validation effort as it was a verification effort. The emphasis of the peer review for verification purposes was to assure that the actual software code faithfully represented the intended methodology for the phenomena being modeled.

3.1.1.7. Model Interactions. ModSAF is a hybrid type of model in that it has characteristics of both an interactive war game as well as a fully automated combat simulation. In the DIS world interoperability is a necessary requirement for all models used in a "Simworld". ModSAF must interact with simulators, treating the simulator as if it were an entity within and belonging to its own set of automated entities. Initial verification tested model operability by simulating the simulator through a local network. Correct Protocol Data Units (PDU's) must pass both in and out of ModSAF. The emphasis was in testing the results of those PDU's received in the ModSAF. The interactive screens on the ModSAF equipment were used to visually verify correctness of responses.

3.1.1.8. User Interfaces. The presentation to the user is an important aspect of a SAFOR. Ease of operation, intuitive actions, readable icons, reduced steps to accomplish an action and other human behavior issues are all things that deserve an expert review by a human factors expert. TRAC-WSMR included user interface as a verification item.

3.1.1.9. Model Responsiveness. It is imperative that the model be able to respond in a real-time mode in order to effect a realistic simulation which involves man-in-the-loop actions. This is a function of both network responsiveness as well as model design. Since the network was fixed and not available during initial V&V, timing tests involved only turn around response time from message receipt until message response (i.e. only internal model actions were considered).

3.1.2 Validation Plan

The validation plan read much shorter than its verification counterpart. Wherever practical, extant Field test data, were utilized. No field exercises were conducted to compare ModSAF with the "real world." Theoretical evaluation was conducted using available literature and/or computer code. Subject matter experts were consulted regarding inputs and in face-to-face observation with model developers while conducting structured walk-throughs of the code.

Comparison of ModSAF results against the results of similar scenario runs in the Combined Arms and Support Troops Force-on-Force Evaluation Model (CASTFOREM--the Army's only accredited high resolution force-on-force model) were the primary evaluation by

comparison. ModSAF results were considered "correct" if they compared in a statistical sense. Acceptable bounds for correctness were established, and anomalies were explained or if considered incorrect by a consensus of functional area experts, code was modified to correct.

3.1.3 Implementation

Verification and validation required access to the ModSAF system, and a suite was installed on-site at TRAC-WSMR. This physical access to the system greatly facilitated both the breadth and depth with which evaluations were conducted. A total of six professional staff years were programmed over two fiscal years for the entire ModSAF V&V effort.

3.2 CCTT SAF Behaviors V&V

This limited detail, high-end SAF behaviors V&V effort is being conducted in concert with other V&V work as a contribution to the overall verification, validation, and accreditation of the CCTT for use as a training device in support of training exercises. The CCTT is a group of interactively networked simulators and command, control, and communication work stations replicating the vehicles and weapons systems of a mechanized infantry or armor battalion task force and its supporting Combat, Combat Support and Combat Service Support elements. It was designed to provide training to individual crew and unit personnel covering the skills and knowledge of crew through company task force level doctrine for the implementation of combat missions. As a training simulation system, the CCTT is expected to contribute by providing a realistic environment in which the necessary enhancement and maintenance of soldier skills may be obtained. This implies that the terrain correlation, system latencies, SAF behaviors and interactions, and the overall fidelity are required to be at such levels that negative training, cartoonish effects, and unrealistic situations are eliminated.

3.2.1 Verification Plan

The process of verifying the SAF behaviors for CCTT regarded making a determination that the model accurately represents the developer's conceptual description and specifications. The main thrust of the developer's concept for SAF behaviors in CCTT concerned implementation of the CISs, that is, the tactics or behaviors exhibited by the SAF were to be those outlined by the CIS.

A critical subset of 276 CISs from the total list of over 700 CISs had been selected by PM CCTT to serve as the baseline for implementation in CCTT. The verification process for CCTT SAF behaviors comprised two interrelated functions regarding this baseline CIS subset. The first of these functions was a determination that the selected CIS subset was in fact available for execution in the final CCTT configuration delivered to the government, and the second comprised activities necessary to assess adequacy of the exhibited behaviors in light of the CIS description.

The methodology necessary to support the first CIS verification function entailed comparing the prioritized baseline subset of the overall CIS list to the CIS available for execution in an exercise. The second CIS verification function entailed scripting a set of scenarios for execution with the SAF and CGF workstations. A sample of behaviors from the baseline CIS subset were selected for assessment, and small vignettes or scenarios requiring execution of each sampled CIS were built and run. Sampling criteria included CISs which enabled assessment of the four measures of effectiveness 1) move, 2) shoot, 3) see, and 4) communicate, and those CISs with disconnect and discrepancy attributes identified as described in the validation section which follows (3.2.2).

Vignettes were prepared to evaluate the CIS-based behaviors in isolation as well as in more complex scenarios which required execution of one or more situational interrupts to other CIS behaviors. Scenarios which were developed encompassed a wide variety of environmental conditions addressing maneuvers and engagements under both day and night conditions. The actual evaluation process for each selected CIS comprised rating each subtask within the CIS on a four-point nominal scale of Approved (A), Needs Improvement (NI), Unacceptable (U), and Not Tested (NT). A log of these isolated and combined scenario executions was maintained, along with the final state of nominal evaluation for each tested CIS subtask, and these results were then documented for inclusion in the final report.

3.2.2 Validation Plan

The process of validating the SAF behaviors for CCTT regarded making a determination of the extent to which the CCTT represents the real world. The unit behaviors in CCTT are

generated from the CISs, and each of the elements of each CIS are validated by the responsible U.S. Army authority before they are accepted for posting into the list. Since the CIS are validated as adequate models for SAF behavior, the only remaining step toward validating CIS-based SAF behavior was assurance that the CIS had been faithfully implemented.

The CIS validation issue necessitated a detailed analysis of each CIS to be implemented in search of situational interrupt target behaviors. Since the CIS themselves have been validated, the validation function served by this effort was to validate the process of turning CIS into software coded behaviors. The methodology entailed tracing each of the 276 baseline CISs in search of the branching destinations on situational interrupts. Each CIS in the baseline subset was analyzed to identify both the conditions requiring a branch to some other CIS, and which specific CIS was the destination for the interrupt. Potential disconnects and discrepancies (such as a closed loop or a branch to a nonimplemented CIS) in the baseline CIS subset were documented and a log of these disconnects and discrepancies and their resolutions were included in the final report.

3.2.3 Implementation

Validation under this methodology was a paperwork exercise, and was completed at TRAC-WSMR with only a copy of the CATT Task Data Base and a listing of the 276 baseline CISs. Verification required access to the CCTT SAF system, and since this could only be accomplished on-site at the development facility in Orlando, it was. A total of three professional staff years were programmed for this entire V&V effort, with 2 people on intermittent TDY over a period of 6 months for the on-site verification segment.

4. Additional CIS Applications

That the CIS is a doctrinally sound description of the necessary combat behaviors make it an invaluable tool for the software developer. The process of turning the natural language CIS into software code has been summarized in an excellent paper by Ourston, et.al. (1995).

In addition to its use as a blueprint for the code development process, the CISs consolidated in the CATT Task Data Base can serve as a tool to scope the overall SAF behaviors development

effort. As described under CCTT validation above, identification of all of the necessary behavioral branches early on in a developmental effort can be of significant value in terms of determining the level of effort necessary to achieve a desired level of functionality.

The CIS subtasks may also serve to function as a sort of managerial checklist. After breaking each CIS into its subtasks, a list of the primitive function calls and code segments begins to emerge. These subtasks may then be cross referenced to parent CIS, and development prioritized in accordance with current management objectives. Thus, the CIS can serve as both the technical breakdown of the work to be performed, and as the ordering plan by which the primitive segments are built up into a fully functional SAF.

5. References

- Denney, Carrol R. (1994). "Modular Semi-Automated Forces (ModSAF) Verification, Validation, and Accreditation Plan," TRAC-WSMR-TR-94-009(R), White Sands Missile Range, NM, 12 pages.
- LORAL/Army Integrated Development Team (1995). "Development of Combat Instruction Sets (CIS) in Support of Doctrine-Based Software," 1246I Research Parkway, Orlando, FL, 34 pages.
- McEnany, Brian R. and Marshall, Henry (1994). "CCTT SAF Functional Analysis," White Paper #178, U.S. Army Program Manager - Combined Arms Tactical Trainer, 12350 Research Parkway, Orlando, FL, 13 pages.
- Mullis, Charles W. (1996). "Close Combat Tactical Trainer (CCTT) Semi Automated Force (SAF) Behavior Verification and Validation (V&V) Plan," TRAC-WSMR-TR-95-036(R), White Sands Missile Range, NM, 12 pages.
- Ourston, Dirk, Blanchard, David, Chandler, Edward, Loh, Elsie, and Marshall, Henry (1995). "From CIS to Software," Proceedings of the Fifth Conference on Computer Generated Forces and Behavioral Representation, IST-TR-95-04, Institute for Simulation and Training, Orlando, FL, Pages 275-285.
- U.S. Army (1993). "Verification, Validation, and Accreditation of Army Models and

Simulations,” DA Pamphlet 5-11, Headquarters, Department of the Army, Washington, DC, 27 pages.

6. Authors' Biographies

Damon D. Baker is an Electrical Engineer in the Model Support Division of the Brigade Modeling Directorate at the U.S. Army TRADOC Analysis Center - White Sands Missile Range. Mr. Baker has a Bachelor of Engineering degree in Electrical Engineering from New Mexico State University. His research interests include semi-automated forces, analysis of distributed simulations, and digital communication network performance.

Charles W. (Chad) Mullis is an Operations Research Analyst in the Model Support Division of the Brigade Modeling Directorate at the U.S. Army TRADOC Analysis Center - White Sands Missile Range. Mr. Mullis holds a Master of Science degree in Geography from the University of North Dakota. His research interests include semi-automated forces, analysis of distributed simulations, and digital communication network performance.

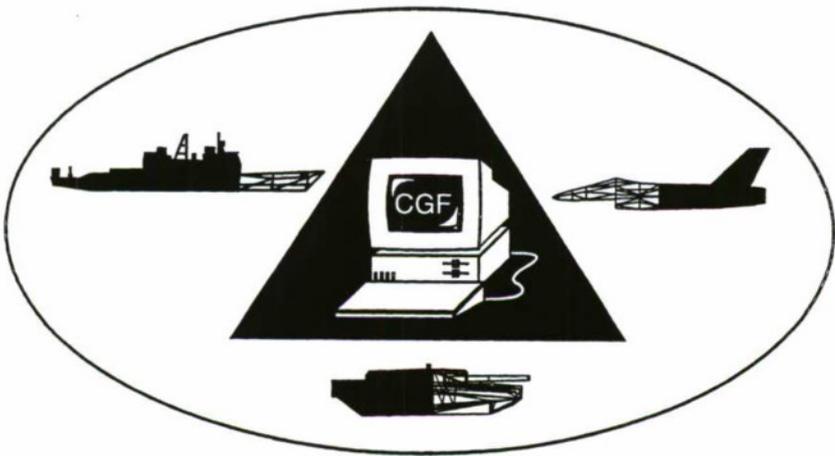
Session 6a: Physical Modeling

Albright, U. S. Army, TRAC, Ft. Leavenworth

Howells, CAE, Canada

Kwak, SAIC

Reece, UCF/IST



Acoustics in Computer Generated Forces

Robert L Albright
TRAC-OAC
ATTN ATRC FM MR ALBRIGHT
255 SEDGEWICK AVENUE
FORT LEAVENWORTH KS 66027-2345
albrihr@trac.army.mil

1. Abstract

Existing systems of Computer Generated Forces (CGF) do not model sound. The need for sound representation became evident in a scenario for demonstration of a research project (Craft 1995) conducted by the Institute of Simulation and Training (IST). An opposing force of a BMP-2 platoon with dismounted infantry are atop a hill occupying defilade positions. They are placed as a strategic road block defending route access to a SCUD launcher. The opposing force should have moved into a hasty defensive position in preparation for an ambush upon hearing sound cues from an approaching United States Marine Corps Advanced Amphibious Assault Vehicle platoon on the other side of the hill. Since Modular Semi-Automated Forces (ModSAF) did not model sound, IST constructed a work-around to simulate the situational awareness cues which would trigger the reaction from the opposing force.

This paper presents a CGF sound model. It addresses production, propagation, and detection of sound waves. Production is the creation of a sound wave, propagation is its spreading, and detection is its reception and perception. Sound waves emitted from vehicles and aircraft are supported.

The model is a physical model with components allowing linkage of behavioral models. Sound production and propagation are the physical models using equations. Detection is the opening where behavioral models can be plugged in. Examples are reactive behaviors to sound cues and the effects of battle sounds on infantry. A reactive behavior to sound cues is exhibited in the scenario above where sound triggers a task transition.

Implementation is in ModSAF and uses the technique of "line-of-sound" to demonstrate linkage of the behavioral to the physical. This modeling process answers "Can this entity be heard?" much like "line-of-sight" for the engagement process answers "Can this entity be seen?" A scenario is played using

sound cues as a trigger for task transition. Two Blue Infantry Javelin teams are hiding behind a tree line in enemy territory awaiting orders for their next move. Meanwhile, a platoon of Red T72M tanks on patrol are moving towards the tree line. A terrain feature blocks line-of-sight. Upon hearing the sound of the approaching vehicles, the BLUEFOR move into a hasty defensive position. They fire upon the tanks after identifying them as enemy.

2. Introduction

The need for sound representation in CGF became evident in a feasibility experiment conducted for the Marine Corps by IST. A scenario created for experimentation showed an opposing force of a BMP-2 platoon with dismounted infantry atop a hill occupying defilade positions. They were placed as a strategic road block defending route access to a SCUD launcher. The opposing force should have moved into a hasty defensive position in preparation for an ambush upon hearing sound cues from an approaching United States Marine Corps Advanced Amphibious Assault Vehicle platoon on the other side of the hill. Since ModSAF, the virtual simulation used for experimentation, did not model sound, IST constructed a work-around that simulated the situational awareness cues to trigger the reaction of the opposing force (Craft et. al. 1995).

The CGF community is faced with the challenge of modeling behaviors as they occur in reality, because behavioral replication of the real-world battlefield is required in CGF (Crooks et. al. 1995). Acoustics, the science of sound (Blitz et. al. 1964), can be modeled and offers progress towards meeting this challenge.

3. Sculpturing Acoustics into CGF

Sequential steps taken to build an acoustics model into CGF were the following: researching the physics of sound, researching how soldiers react to sound cues, linking the behavioral to the physical, choosing a reactive behavior for demonstration of its

2. Design

In order to support Navy and Marine Synthetic Forces simulations (Tracor, 1996), the Compact Terrain Database (CTDB) used by ModSAF has been expanded to include a representation of the ocean floor along with a more complete representation of the ocean surface. Table 1 shows the various ocean characteristics that are represented.

The ocean bottom is represented using the existing terrain representation (i.e., grids, TINs, and microterrain) (Stanzione, et. al. 1996), and additional supported soil types. Many ocean “features” are really abstractions describing pieces of the terrain. As such, the physical representation of such features can be adequately handled by incorporating their structure into the polygonal representation of the ocean bottom. Abstract notions such as “this area of the terrain is a reef” can be explicitly stored as abstract features using existing CTDB mechanisms.

The representation supports tidal variation of the ocean surface. In the coastal regions, the absolute elevation of the water’s surface is specified, subject to some maximum x-y bounds. Within the specified region, any area where the water elevation exceeds the land elevation is covered by water, and any area

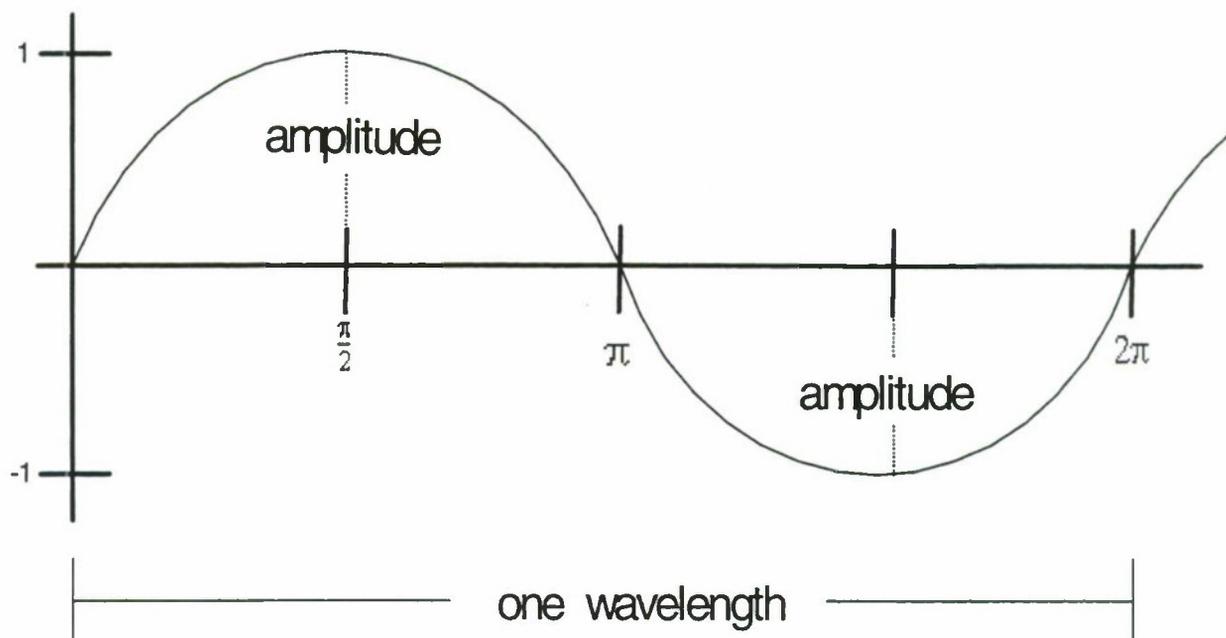
where this is not the case is dry (or perhaps moist) land, as shown in Figure 1. The bounding polygons for water bodies are defined by high tide position, so that the surface elevation can be decreased to represent lower tide levels. In order to allow for changing tides in real-time, all water polygons reference a tidal zone. Each zone stores an offset, which is added to the surface elevation stored for the polygon. Thus, changing the tide in a region is simply a matter of changing that region’s tidal offset.

In most areas, the ocean surface is represented by single square polygons that correspond to the size of a CTDB terrain patch. The representation can not be too coarse because in databases that use the Global Coordinate System (GCS) (Evans, 1995) the ocean surface is curved. On the other hand, the representation can not be too fine or it will use much more memory. The patch size is the largest size at which integration into existing intervisibility algorithms is straight forward, since the intervisibility code already performs a patch traversal. The representation consists of a single elevation value for water in the patch, and a reference to additional surface characteristic data. It is assumed that there will be few unique sets of surface characteristics relative to the number of patches.

Table 1: Ocean Representation Characteristics

	Multiple Elevation Surfaces	Advanced Features and Attributes	Dynamic Terrain
Ocean Floor	<ul style="list-style-type: none"> ◆ Bathymetry data ◆ Extended soil types infrastructure to include bottom characteristics 		
Ocean Surface	<ul style="list-style-type: none"> ◆ Patch and Wet TIN surface ◆ Sea State attributes (primary and secondary wave height, period, speed, direction) ◆ Surface Temperature 		<ul style="list-style-type: none"> ◆ Dynamic sea state and surface temperature
Surf Zone	<ul style="list-style-type: none"> ◆ Tidal Zone with offset for surf height 	<ul style="list-style-type: none"> ◆ Man made features (wharves, piers, etc.) 	<ul style="list-style-type: none"> ◆ Variable tidal zone offset
Rivers	<ul style="list-style-type: none"> ◆ Wet TIN surface 		

$$y = \sin(x)$$



Graph 4.1

linkage to the physical, implementing the model in a CGF system, and testing its validity.

4. Background

4.1 Physical Research

A disturbance generated from a physical, as opposed to chemical, change by means of applied force will input energy into the medium surrounding it. This causes the immediate surrounding particles to undergo back and forth motions about their equilibrium positions. These particles influence their neighboring particles to behave likewise as the energy carries the disturbance particle to particle, away from its source. The disturbance travels in the form of a wave and has a distinct wave motion (Blatt et. al. 1989, Boleman et. al. 1989).

Graphically, a wave is described by the sine function (Boleman et. al. 1989). Graph 4.1 illustrates y as a function of the sine of x . The sine curve this function produces represents the behavior of one particle in the medium when it encounters the passing disturbance (Whitten et. al. 1996).

One wavelength is an observed single back and forth motion of the individual particle. One complete up-

and-back (makes one "hill"), then down-and-back (makes one "valley") motion is an oscillation. The number of oscillations a particle will undertake is a function of the amount of energy given off by the disturbance. A greater amount of energy produces more oscillations. The number of oscillations per second is the frequency of the wave (Boleman et. al. 1989).

Amplitude is the maximum distance an individual particle in the medium will move about its equilibrium position as the disturbance passes. This wave characteristic is used to gain a measure of the amount of energy given off by the disturbance (Sears et. al. 1987).

A sound wave originates from a vibrating source producing compressional wave motion in the surrounding medium. The vibrations shove the particles in its way closer to their surrounding particles, leaving a partial vacuum in the wake of their movement. The medium around the partial vacuum rushes in from all sides due to pressure from

the compression pushing them back. Alternating regions of particles compressed together and spread apart occur along the distance the disturbance travels. These pressure fluctuations spread in all directions away from the source. As the distance from the source increases, the flow rate of the disturbance diminishes because the amount of energy carrying the disturbance becomes less, and the amount of energy given off as heat, due to friction, increases. This is the property of attenuation (Blatt et. al. 1989, Blitz et. al. 1964, Boleman et. al. 1989).

4.1.1 Acoustical Phenomenon

Inspecting a wave's life cycle unveils three credentials of acoustics. They are a sound wave's production, propagation, and detection. Production is the sound wave's creation, propagation is its spreading through a medium, and detection is its reception and perception (Blitz et. al. 1964).

4.2 Behavioral Research

Detection provides the opportunity for a sound stimulus to trigger a reaction when the stimulus is processed (Boleman et. al. 1989). A reaction of interest to a researcher may be internal or external. Internally, the heart rate of a soldier may increase at extreme sounds that suddenly occur from ammunition detonation in combat. Externally, a sound cue will alert a soldier and establish situational awareness (Mullally et. al. 1995). For example, a soldier at rest may become hastily defensive upon hearing a sound cue in the distance.

Psychoacoustics is the study of psychological interactions between humans and the world of sound. The primary perceptual attributes of sound are loudness and pitch (Parker et. al. 1988). The information describing them are transmitted in a propagating sound wave. Loudness, or a sound's intensity level, is a measure of how much energy per second the wave brings next to the detection device (Blatt et. al. 1989). It depends on a sound's physical

intensity, amplitude, and frequency. Pitch is the highness or lowness of a sound and depends on a sound's frequency (Parker et. al. 1988).

5. Linkage of Behavioral to the Physical

The three credentials of acoustical phenomena suggest three platforms in the acoustics model which aid in tying all the credentials together to produce an acoustics model which can be implemented. The platforms are the physical, behavioral, and linkage. The physical platform scientifically represents sound and addresses its production and propagation. The behavioral platform addresses reactivity to sound cues in the environment. Finally, the linkage platform connects the behavioral to the physical with sound detection being the opening where behavioral models plug in. The three platforms are wound together in a shape that gives a form the acoustics model can be implemented.

6. Implementation

The acoustics model is implemented in the CGF simulation ModSAF, version 2.0. Five physical models, one behavioral model, and one linkage model are implemented. The physical models are sound velocity, intensity, intensity level, power level, and pressure level. The behavioral model is task transition upon hearing sound cues. The linkage model is line-of-sound.

6.1 Sound Velocity Model

The sound velocity model returns the speed of sound in a gas. Sound velocity is a function of air mass, humidity, and temperature.

Gas	% Composition
Nitrogen (N2)	78.09
Oxygen (O2)	20.95
Water vapor (H2O)	0-4
Argon (Ar)	0.93
Carbon dioxide (CO2)	0.03
Neon (Ne)	0.0018
Helium (He)	0.000524
Krypton (Kr)	0.0001
Radon (Rn)	6 x 10-18

Table 6.1.1.1

Equation 6.1.1 drives this model and gives the velocity of a sound wave in meters per second (Blatt et. al. 1989).

$$v = \sqrt{\frac{\gamma k T}{m}}$$

Equation 6.1.1

T is the temperature of the gas expressed in Kelvins. K is Boltzmann's constant. Gamma is the ratio of specific heat (Blatt et. al. 1989).

Specific heat is the amount of heat that must be supplied to an object to effect a temperature change. The ratio of specific heat is the ratio of the specific heat of a gas at constant pressure to its specific heat at constant volume (Blatt et. al. 1989).

M is the molecular mass of the gas serving as the medium for the traveling sound wave. The experiment used for demonstration of the acoustics model propagates sound through air. Air is a mixture of gases and does not exist as a single molecular structure, therefore its mass cannot be measured (Boleman et. al. 1989).

6.1.1 Virtual Air

Virtual air was created to resolve this conflict. It portrays air as having a single molecular structure while maintaining its percent composition as a mixture. The percentage of the gas that occurs in air is the percentage of gas characteristic in one virtual air molecule. Consider a volume of one-hundred molecules. This volume is a mixture of four gases. Ten molecules are carbon, twenty argon, thirty oxygen, and forty nitrogen. The volume of gas is then 10% carbon, 20% argon, 30% oxygen, and 40% nitrogen. The virtual molecule representation also

maintains these percentages. One virtual molecule would be comprised of 1/10 carbon, 1/5 argon, 3/10 oxygen, and 2/5 nitrogen. One hundred virtual molecules placed in the same volume will maintain the same percentages. 1/10 carbon multiplied by 100 virtual molecules is 10% carbon, 1/5 argon multiplied by 100 virtual molecules is 20% argon, 3/10 oxygen multiplied by 100 virtual molecules is 30% oxygen, and 2/5 nitrogen multiplied by 100 virtual molecules is 40% nitrogen. Table 6.1.1.1 shows the gases that comprise air and their percentage of occurrence (Boleman et. al. 1989).

6.2 Sound Intensity Model

The sound intensity model returns a measure of the sound energy radiating from a source in all directions. Sound intensity is a function of sound power emitted from a source.

The intensity of a sound wave is defined by the energy it brings into a unit of area each second. It is calculated using equation 6.2.1 (Blatt et. al. 1989).

$$I_r = \frac{P}{4\pi R^2}$$

Equation 6.2.1

I subscript r is expressed in watts per meter squared, P is the sound power expressed in watts, R is the range (in meters) between source and listener, and 4*PI*R accounts for the source radiating sound energy uniformly in all directions (Blatt et. al. 1989).

6.3 Sound Intensity Level Model

The sound intensity level model returns the loudness of a sound source at the listener. Sound intensity

level is a function of the source's sound intensity. It is calculated using equation 6.3.1 (Blatt et. al. 1989).

$$\beta = 10 \log_{10} \frac{I_r}{I_0}$$

Equation 6.3.1

Beta is expressed in decibels (dB). Sound intensities are naturally compared by humans logarithmically, therefore their levels are specified on a logarithmic scale. Two sounds differ in intensity by one bel if the ratio of their intensities is ten. The decibel, one-tenth of a bell, is a more common expression used (Blatt et. al. 1989).

I subscript r is the sound intensity of a source perceived by a listener at a given range. I subscript zero is the sound intensity at the threshold of audibility (Blatt et. al. 1989).

6.4 Sound Power Level Model

The sound power level model returns the sound strength of a source. Sound power level is a function of the sound power emitted from a source. Sound power is the rate at which sound energy is spread. Sound power level is calculated using equation 6.4.1 (Besancon et. al. 1966).

$$PWL = 10 \log_{10} \frac{W}{W_{ref}}$$

Equation 6.4.1

W is the sound power expressed in watts. W subscript ref is the sound power at the threshold of audibility (Besancon et. al. 1966).

6.5 Sound Pressure Level Model

The sound pressure level model also returns the sound strength of a source. Sound pressure level is a function of the density of the gaseous medium a sound wave is propagating through. It is calculated using equation 6.5.1 (Besancon et. al. 1966).

$$SPL = 20 \log_{10} \frac{p}{p_{ref}}$$

Equation 6.5.1

Rho is the density of air. Rho subscript ref is the density of air at the threshold of audibility (Besancon et. al. 1966).

6.6 Task Transition Upon Hearing Sound Cues

This behavioral model supports alertness and situational awareness. It exists in ModSAF as an enabling task (libesound) and is easily detached when the level of sophistication this behavior model offers is not needed.

6.7 Line-Of-Sound

Line-Of-Sound propagates a sound wave between a source and listener to answer "Can this entity be heard?" The listener has an intensity level of background noise occurring around him or her. A sound source in the distance will be heard if the intensity level of its sound wave is greater than the intensity level of the background noise.

Four combinations of source to listener can occur. They are individual to individual, individual to group, group to individual, and group to group, with the first two being supported, respectively.

6.8 Libsound

ModSAF's libsound was created to contain the physical and linkage models. It can be easily detached when the sophistication an acoustics model offers is not needed.

7. Experimentation

A scenario was created to demonstrate the acoustics model. Two Blue Infantry Javelin teams are hiding behind a tree line in enemy territory awaiting orders for their next move. Meanwhile, a platoon of Red T72M tanks on patrol are moving towards the tree line. A terrain feature blocks line-of-sight. Upon hearing the sound of the approaching vehicles, the BLUEFOR should move into a hasty defensive position and fire upon the tanks after identifying them as enemy.

8. Results

The Blue Infantry Javelin team moved from a state of rest to hasty defensive position upon hearing the sound cues of approaching vehicles. After identifying the approaching vehicles as enemy, they fired upon the REDFOR.

9. Conclusion

Sound is represented in a CGF system. Its representation demonstrates progress towards realism in behaviors by entities on the virtual battlefield. Acoustics in a CGF simulation adds the sophistication to behaviors that the CGF community assumes to be present.

10. Future Work

Behaviorally, the acoustics model could be used to study effects of battle sounds on infantry. Support for sounds of weapons fire and ammunition detonation would need to be included to support this. Vehicle identification by sound signature could also be incorporated. Pitch would need to be modeled to support this.

Physically, the acoustics model could be extended. Solid mediums could be included in sound propagation. Phenomenons affecting sound propagation, such as reflection, absorption, interference, turbulence, and refraction (turning) could be incorporated.

This acoustics model only supports sounds emitted from an individual vehicle. Sound blending caused by units of more than one vehicle is also needed.

The concept of "virtual air" assumes that the dispersion of the gases comprising air is evenly distributed and no concentration of one gas hovers nap of the earth. Validation is needed that compares sound velocity calculated in "virtual air" against sound velocities recorded in air.

11. Acknowledgments

The author thanks Robert Franceschini, Dan Mullally, Kent Pickett, Derrick Franceschini, Matt Kraus, Jimi Whitten, IST, TACOM and AMSAA for their technical support on this experiment conducted at IST during development training as an intern. An additional thank you is extended to Robert Franceschini and Kent Pickett for editorial support.

12. References

- Besancon, Robert M. (1966). *The Encyclopedia of Physics*, Reinhold Publishing Corporation, New York, pp. 31-32.
- Blatt, Frank J. (1989). *Principles of Physics, Third Edition*, University of Vermont, Allyn and Bacon, pp. 269, 291-313, 338-414.

- Blitz, J. (1964). *Elements of Acoustics*, Butterworth & Co. (Publishers) Ltd., p. 1.
- Bolemon, Jay (1989). *Physics An Introduction*, Second Edition, Prentice-Hall, Inc., pp. 299-319, 340-359.
- Craft, Michael A., Franceschini, Derrick J., Kraus, Matthew K., Mullally, Daniel E., Adkins, Michael K., Albright, Robert L., Nida, Jonathan C., and Napravnik, Lee J. (1995). *AAAV: Demonstrating the Feasibility of Using Virtual Simulation for Test and Evaluation*. Contact Number #N61339-92-K-0007 A0003, Institute for Simulation and Training, University of Central Florida.
- Crooks, William H., Ph.D. (1993). "A Command and Control Metaphor For Computer-Generated Forces", *Proceedings of the Third Conference on Computer Generated Forces and Behavioral Representation*, Institute for Simulation and Training, Orlando, Florida, p. 395.
- Mullally, Daniel (Subject Matter Expert). (1995). Personal interviews. Orlando, Florida: Institute for Simulation and Training.
- Parker, Sybil P. (1988). *Acoustics Source Book*, McGraw-Hill, Inc., pp. 1-23, 270-330.
- Pickett, H. Kent (Director of Modeling and Research). (1995). Telephone interview. Fort Leavenworth, Kansas: TRADOC Analysis Center.
- Sears, Zemansky, Young. (1987). *University Physics*, Seventh Edition, Addison-Wesley Publishing Company.
- Whitten, Jimi (Operations Research Analyst). (1996). Personal interview. Fort Leavenworth, Kansas: TRADOC Analysis Center.

13. Author's Biography

Robert L. Albright is a Computer Scientist for the Modeling and Research Directorate at TRADOC Analysis Center in Fort Leavenworth, Kansas. Mr. Albright has a Bachelor's of Science Degree in Computer Science and is working towards a Master's of Science in Operations Research. His interests are modeling CGF at the virtual level and computer graphics.



Creating a Synthetic Environment for Naval Applications

P.B. Howells, Ph.D, G. Giguere, Eng
CAE Electronics Ltd.
St-Laurent, Quebec, Canada

1. Abstract

The use of synthetic environments in the areas of training, research and development, and equipment evaluation is becoming common practice. This paper will describe an existing simulation package developed for air force and army applications that has been expanded to cater for Anti-submarine Warfare (ASW), Anti-surface Warfare (ASuW), Search and Rescue, and Naval Gunfire Support. It will give an overview of the existing simulation package and a description of the models that have been developed to support a naval environment. A scenario in which a helicopter prosecutes a submarine is presented as an example problem to illustrate the application of the new software.

2. Introduction

The generation of synthetic environments for naval applications is concerned with the simulation of entities such as submarines, surface ships, aircraft and helicopters. Typical weapons and sensors carried by these platforms may include missiles, rockets, bombs, torpedoes, mines, radars, passive sonars, and active sonars. The scenarios that can be created using these elements encompass Anti-submarine Warfare (ASW), Anti-surface Warfare (ASuW), Search and Rescue and Naval Gunfire Support.

CAE's Interactive Tactical Environment Management System (ITEMS) has been expanded to include the above platform types, weapons and sensors. The models developed are physical models that consider the essential parameters that affect performance. ITEMS uses an off-line Database Management System (DBMS) for the player definition and scenario creation. The DBMS user interface provides the user with the flexibility to modify and create new platform types and new training scenarios.

This paper gives details of the models used to describe the above platform types, weapons and sensors. Details are also given of manoeuvres such as

zig-zag patterns for ships and dip manoeuvres for

helicopters. A scenario in which a torpedo armed helicopter prosecutes a submarine is presented as an application problem. Such a scenario could be created to support a training exercise, or an equipment evaluation process where ITEMS acts as the target generator.

ITEMS includes many more platform types, weapons and sensors than those mentioned above for army and air force applications. The present paper however will be confined to the naval elements. Since all three services are supported, ITEMS lends itself to combined operations.

3. ITEMS Overview

CAE's Interactive Tactical Environment Management System (ITEMS) provides the synthetic environment for air, land and sea environments. ITEMS is used in a number of research and training facilities worldwide to support training and equipment evaluation.

The basic element within ITEMS is the Player. A Player is defined as anything of tactical importance: an aircraft, a tank, a ship, a submarine, a surface to air missile installation are just a few examples. Each Player may be assigned a range of weapons and sensors.

Scenarios are defined off-line using the DataBase Management System (DBMS) which use low-level databases that define the weapons, sensors, platform dynamics, and behaviour. Figure.1 illustrates the setup. DBMS comprises a specialised set of editors based on the X-windowing system.

An important part of Player definition is the modelling of behaviour. ITEMS uses a rule based system that allows Players to determine their opponent and how to react. In the case of a ship moving through hostile waters, the knowledge based rule set could, for example, control the ship to

perform a zig-zag pattern and set all sonars to passive mode. The same rule set could force the ship to deploy countermeasures if it detects an incoming torpedo.

As an aid to scenario creation and scenario execution, ITEMS uses a map display and a stealth view.

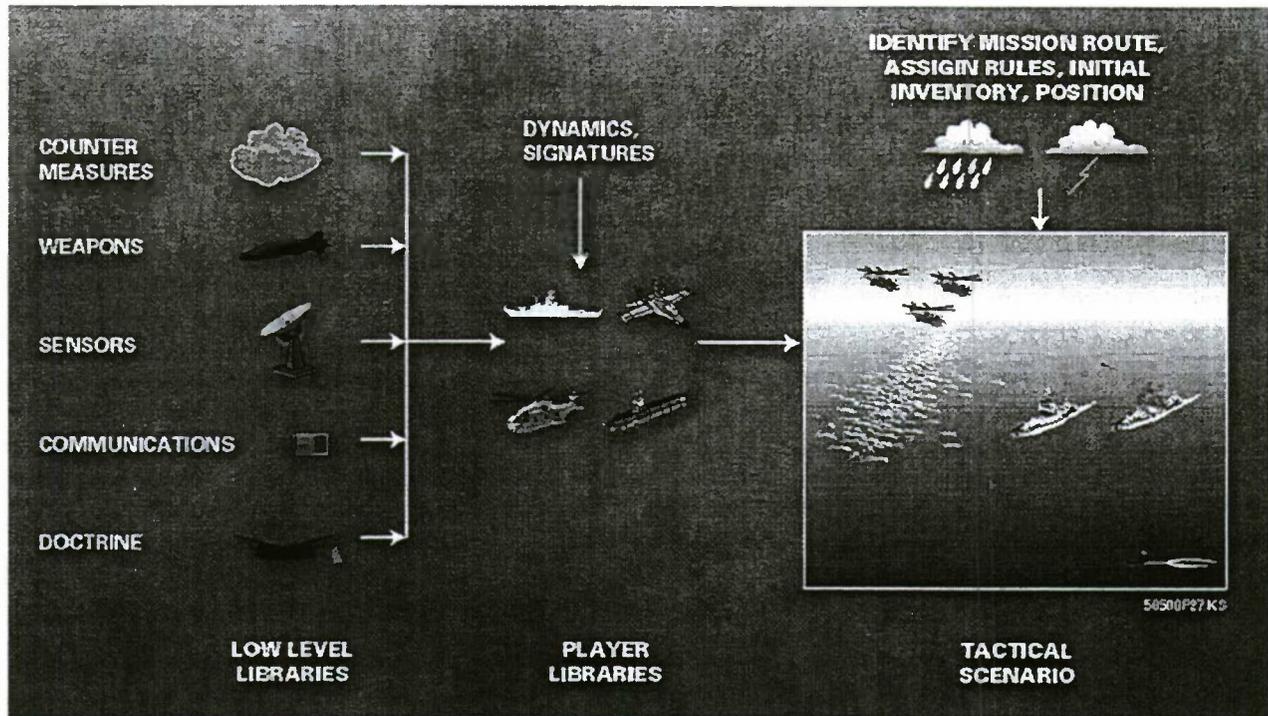


Figure 1: ITEMS Architecture

4. System Models

The following section gives an overview of the mathematical models developed to describe the above mentioned platform types, weapons and sensors.

Active Sonar:

Illustrated in Figure.2, are the principal factors that affect the target detection process in the case of an active sonar: the source level (SL), transmission loss (TL), target strength (TS), the noise level (NL), directivity index (DI), reverberation level (RL), and the signal processing gain (PG).

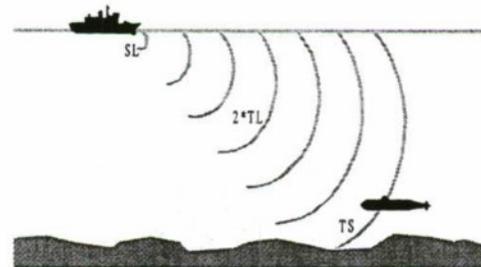


Figure 2: Active Sonar model

Each of the variables are related according to the expression:

$$S/N = S_L - 2T_L + TS - \max(R_L, NL-DI) + P_G$$

where the ratio (S/N) is the signal to noise ratio.

The noise level is calculated from the noise sources defined at the scenario creation time and include ambient and shipping noise. The transmission loss is an important parameter in the underwater detection process and this is calculated from look-up tables entered at the DBMS level. The tables require the

user to define the sound channel which provides for increased detection ranges under certain conditions. The input variables to the look-up tables include range, sonar operating frequency, and whether the sonar platform, or target lie within the sound channel.

The parameter P_G is the signal processing gain. This parameter is defined by the user and represents the increase in the signal to noise ratio that is achievable using advanced signal processing.

A target is detected using either a deterministic model, or a probabilistic model. In the case of the deterministic model, a target is detected if the signal-to-noise ratio exceeds a fixed threshold value. In the case of the probabilistic model the signal-to-noise ratio is used to compute the probability of detection from a probability of detection versus signal-to-noise ratio curve entered via DBMS. A random number is then drawn from a gaussian distribution and compared with the probability of detection value. If the random number is less than or equal to the probability value, then the target is detected. The two models allow the user to define either a deterministic or probabilistic scenario. For example, while training the sonar operator it may be advantageous that target detection be random, where for the equipment evaluation process fixed detection ranges may be desirable to keep the problem tractable.

The active sonar model is used to represent hull mounted sonars for ships and submarines, sonobuoys, and torpedo acoustic homing heads.

Passive Sonar:

The passive sonar model is illustrated in Figure .3 below.

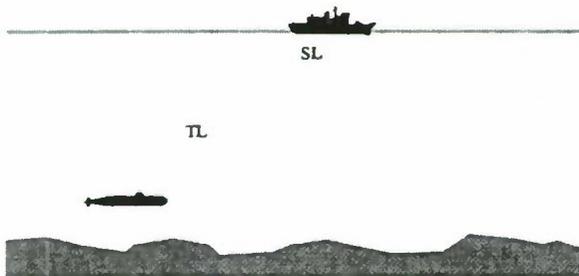


Figure 3: Passive Sonar model

As shown in Figure.3, The variables affecting the passive target detection process are the same as those for the active sonar. The absence of the active

transmission signal, however, means that the signal-to-noise ratio is given by a slightly modified equation:

$$S/N = S_L - T_L - NL + DI + P_G$$

The sources of the different noise terms is the same as for the active sonar.

The passive sonar model is used to model such entities as towed arrays, sonobuoys, passive listening hull mounted sonars and torpedoes.

Acoustic Decoys:

The acoustic decoy model simulates acoustic countermeasures that may be deployed from sea platforms, or towed behind a surface ship. The nixie is an example of the latter. The countermeasure model generates an additional noise term that appears in the sonar equations. The noise level that is calculated takes into account the attenuation of the noise signal as it propagates through the water.

Surface Ships:

Surface ships are represented as linearized dynamic models with an autopilot for speed and course control. A simple model is available to represent the sea state based on four sinusoids generated from the sea spectrum appropriate to the sea state for the scenario. The four sinusoids give rise to the wave height which is calculated at four locations around the ships hull which are then used to generate buoyancy forces and moments that give rise to ship pitch, roll and heave motions

Submarines:

Submarines, like the ships, are modelled as linear dynamic models. An autopilot system is available for speed, course and depth control.

Anti-Ship Missiles:

ITEMS already supports the simulation of a range of Proportional Navigation and Command to Line-of-Sight Missile Systems. For the Naval environment, provision is made for the definition of height and azimuth inertial steering to simulate anti-ship missiles like Harpoon and Sea Eagle. A range of terminal

homing modes from pop-up and dive to terminal homing at sea skimming height are also available. Provision is available for defining a an inertial navigation course so that the missile can fly an off-set way point from the target to create a dog-leg trajectory.

Torpedoes:

The torpedo model is a full five-degree (roll stabilised) freedom model that considers the principal factors that affect performance: thrust developed by the propulsion system, drag and weight. The torpedo may be launched from either submarines, ships, helicopters, or aircraft. A range of search patterns may be defined via DBMS which the torpedo executes to search for the target. The acoustic sensor assigned to the torpedo may operate in either a passive or active mode. Once the torpedo has locked on to the target it is steered on the target using a proportional navigation law. The torpedo remains locked on to the target unless there is a drop in the signal to noise ratio, brought about, say, by the deployment of countermeasures.

Depth Charges:

Depth charges may be deployed from helicopters, surface ships, or aircraft. The model considers the mass of the depth charge, drag coefficient, and initial launch conditions, such as velocity and attitude. When in air, the depth charge follows a parabolic trajectory up to the point where it broaches the waters surface, at which point it sinks at a user specified sink rate. Detonation occurs based on a depth setting entered via DBMS.

Sonobuoys:

Sonobuoys may be deployed from any air platform. Their dynamic characteristic are much the same as the depth charge for the in-air phase. Sonobuoys may be deployed in varied patterns, and may be set to operate in either a passive or active mode.

Manoeuvres:

In order to complement the naval environment entities, automatic manoeuvres have been developed to enhance realism. Such manoeuvres include zig-zag patterns for ships, an auto dip pattern for naval helicopters equipped with a dipping sonar,

manoeuvres for units involved in a screening pattern and generic fixed wing and rotary wing search patterns for submarine detection and prosecution.

5. Example Problem

The purpose of this example is to demonstrate how the ITEMS system makes use of the above entities to create a naval scenario. Space does not allow for a full description of each player and player rule sets, so only a sub-set will be presented here.

5.1 Scenario Description

In Figure.4 are presented details of the scenario. The figure shows a airborne early warning radar equipped aircraft which detects submarine periscope at the surface. The aircraft send a tactical message to a nearby surface ship which dispatches a helicopter equipped with a dipping sonar and torpedo to prosecute the submarine. The submarine detects the incoming helicopter on its radar, and submerges. The helicopter loses contact with the submarine and proceeds to dip its sonar at the last known reference point. The helicopter is successful in locating the submarine and deploys a torpedo. The submarine detects the incoming torpedo and takes evasive action by manoeuvring and deploying counter measures.

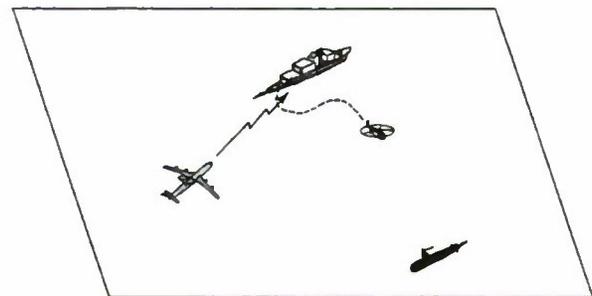


Figure 4:Scenario Overview

5.2 Player Definition

When creating the above scenario, the first step is to define the Player types; in this case the aircraft, ship, helicopter and submarine. The definition of these players requires the user to specify details of the weapons and sensors, typically the radar carried by

the aircraft and the sonar and torpedo carried by the helicopter.

In Figure 5. is illustrated a sample DBMS page for the definition of the dipping sonar. The data fields are shown commensurate with the parameters that appear

in the sonar equation defined above. The table that defines the probability of detection versus range curve is also shown. Similarly pages are available for defining the player characteristics (dimensions and maximum speed for example) and weapon system

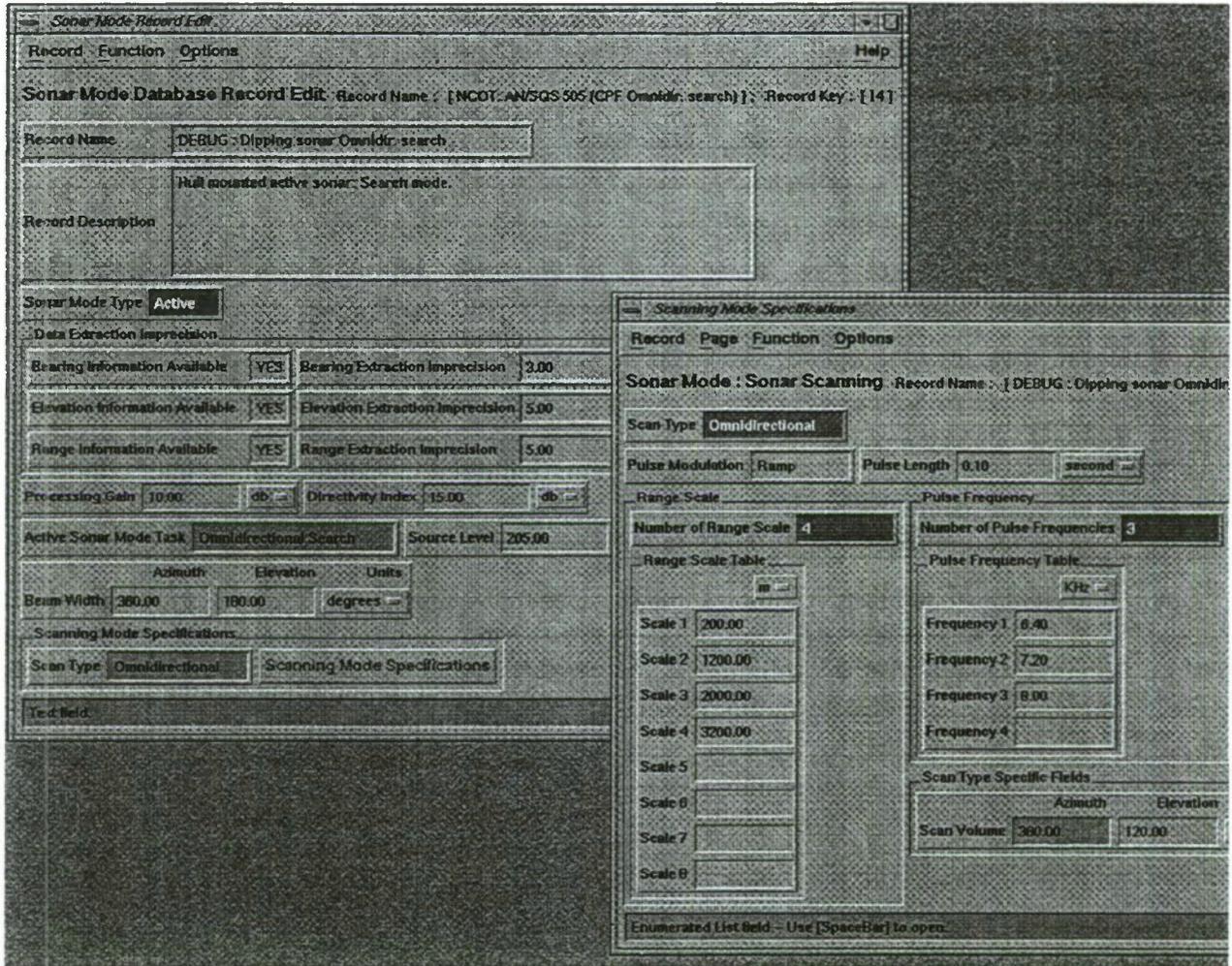


Figure 5: Active Sonar Mode definition

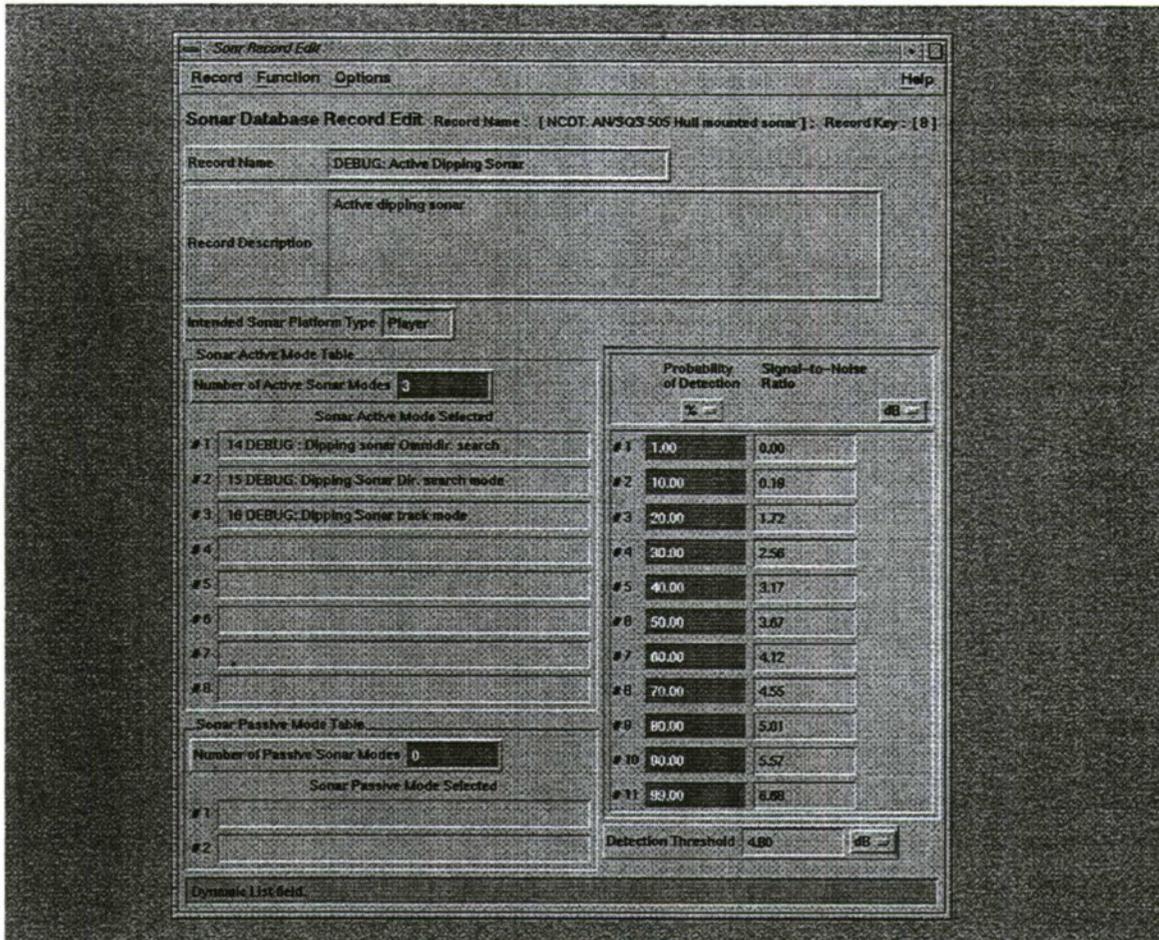


Figure 6: Active Sonar definition

5.3 Rule Sets

As mentioned in section.2 above, player behaviour is represented using an rule based system. A sample rule set for the submarine player is shown in Figure. 5. The rules require the submarine to submerge if it detects an air or surface contact. Further, if the submarine detects an incoming torpedo it is to manoeuvre and deploy countermeasures. As Figure .5. shows, the rules are based on "IF", "THEN" statements that make use of condition and response parameters.

Rule Set Name	RED SUB ACTION RULE	Condition	Response
1.1	IF UNDER ATTACK	FALSE	THEN EXECUTE RESPONSE EXECUTE RESPONSE CHANGE RELATIVE DEPTH = 50 M TURN RADAR ON
1.2	IF PO PLATFORM = AIRBORNE	TRUE	THEN TURN ALL RADAR OFF CHANGE RELATIVE HEIGHT TO DEPTH = 50 M
1.3	IF UNDER TORPEDO ATTACK	TRUE	THEN EXECUTE RESPONSE EXECUTE MANEUVER TRAJECTORY MANEUVER TRAJECTORY NAME TORPEDO EVAASIVE MANEUVER EXECUTE RESPONSE DEPLOY STATIC ACOUSTIC BECDY SET GLOBAL VARIABLE - UNDER ATTACK TRUE
1.4	IF UNDER TORPEDO ATTACK AND GLOBAL VARIABLE UNDER ATTACK AND GLOBAL VARIABLE TIME STAMP UNDER ATTACK > 120 AEC	TRUE	FALSE THEN EXECUTE RESPONSE - GO TO PERISCOPE DEPTH EXECUTE RESPONSE TYPE ON RADAR SET GLOBAL VARIABLE UNDER ATTACK
1.5	IF PO PLATFORM TYPE = SURFACE	FALSE	THEN EXECUTE RESPONSE APPROACH PRIME OPPONENT RD SRG 0.0 DEG REL RSG 500 M REL Z OF FBET -50 M NAV BOA DIRECT
1.6	IF PO PLATFORM TYPE = SURFACE AND RANGE < 100 M	FALSE	THEN EXECUTE RESPONSE FIRE WEAPON TORPEDO

Figure 7: Red Sub Action ruleset

5.4 Scenario Execution

In Figure.6 is shown a snapshot of the Tactical

Situation Display (TSD) that is available for the monitoring the scenario during run time. The Forward View Display (FVD) although available is not shown.

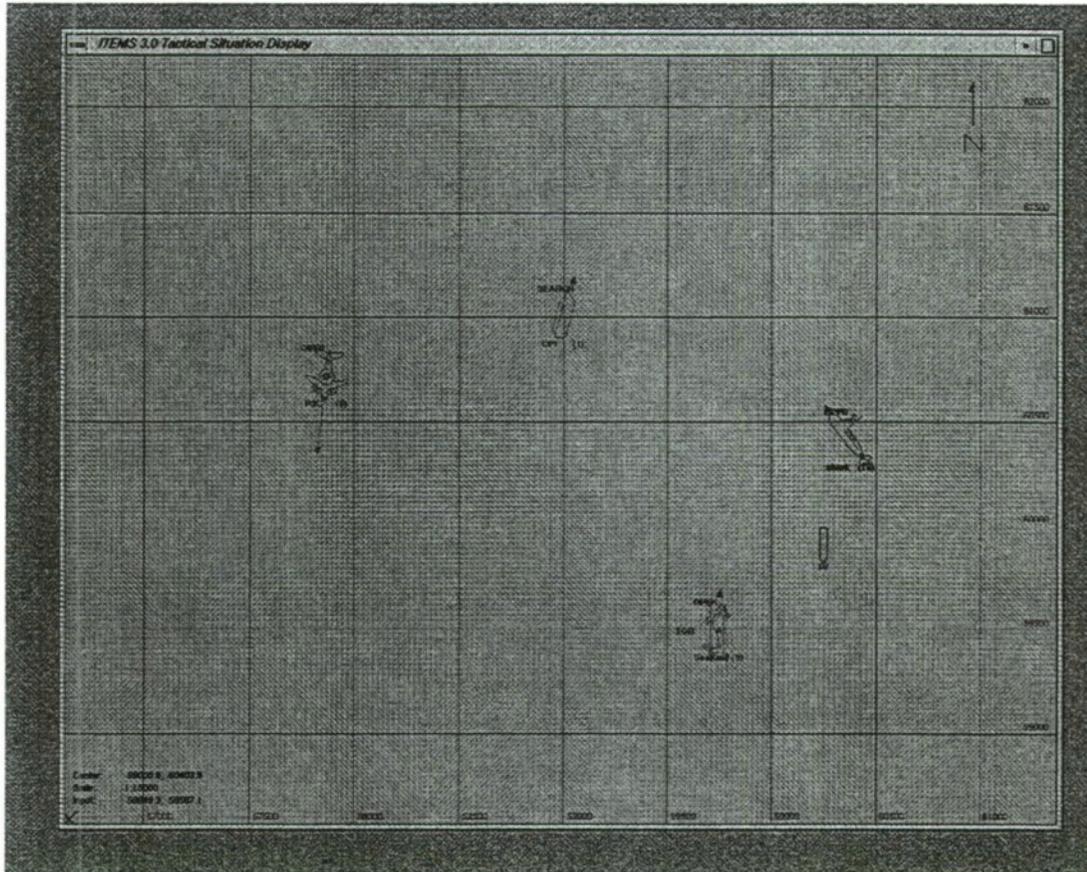


Figure 8: ITEMS Tactical Situation Display

6. Conclusion

Details have been presented of extensions to the ITEMS synthetic environment simulation package to support naval applications. Key to the new development is the use of physical models that consider the principal factors that affect performance. In the case of the acoustic environment, the sonar equation is used in both its active and passive form to simulate the target detection process.

An example problem involving an attack on a submarine was presented to illustrate the use of some of the naval elements that had been developed,

as well as the concept of player definition and the creation of rule sets for the control of players.

7. Acknowledgements

Acknowledgement must be given to members of the Virtual Environment Development team of CAE Electronics whose hard work and vision have made the naval environment possible.

8. Author's Biographies

Dr Peter. B. Howells is the manager of the Virtual Environment Development Department at CAE Electronics, Montreal. He has been involved in the modelling and simulation of physical systems for a number of years while working at various research establishments. His current interest lies in the modelling of computer generated forces for naval applications.

Ghislain. Giguere holds a Bachelor's degree in engineering and is a principal member of the Virtual Environment Development team at CAE Electronics working on the modelling and simulation of naval elements.

Phenomenology Behaviors in ModSAF

Se-Hung Kwak
Lockheed Martin Advanced Distributed Simulation
50 Moulton St., Cambridge, MA 02138
skwak@camb-lads.loral.com

Reba Lyons
US-Army, STRICOM
AMCPM-DIS
12350 Research Parkway, Orlando, FL 32826-3276
lyons@stricom.army.mil

1. Abstract

Phenomenology sensitive behaviors are key components for constructing a realistic CGF. However, research and development of these behaviors have been much delayed because of complex and yet fuzzy nature of the behaviors and because of scarce mathematical models and data. This paper described a new development in ModSAF for phenomenology sensitive behaviors. Behavioral scaling, a novel approach, is taken for the phenomenology behaviors project, which allows infinitive numbers of behavioral variations of a single base behavior by scaling the base behavior depending on environmental conditions. To support this concept, a phenomenology behavior architecture is introduced. It is based on a behavioral taxonomy; passive, reflexive, reactive, and reflective behaviors. In order to facilitate the behavioral upgrade process as well as to improve modularity of the process, environment assessment and reasoning libraries (modules) are also newly built. The behavioral scaling concept, phenomenology behavioral architecture, and environmental assessment and reasoning libraries significantly saved development time and efforts. Moreover, they provide a generic framework for adding new phenomenology sensitive behaviors. This paper also discusses phenomenology (environment) models implemented in ModSAF.

2. Introduction

ModSAF is one of the most widely-used constructive/virtual CGF programs. Since its inception in 1993, it has grown rapidly with the continuous addition of new functionality, facilitated by its modular architecture. One of the recent major advances in ModSAF is the inclusion of phenomenological effects transforming ModSAF's simple, constant, high-noon, clear-day environment to a dynamic environment. Currently, ModSAF supports rain, fog, smoke, dust, and illumination variations due to the locations of the sun and the moon (time-of-day effects). Even though the dynamic nature of phenomenology is supported in ModSAF,

most ModSAF behaviors used to assume the high-noon, clear-day environment that was present at their creation.

Thus, LADS phenomenology behavior (LPB) group started to work to eliminate the limitation. First, the group surveyed to identify the impacts of these phenomenological effects for all behavior libraries in ModSAF. Based on the observations, an architecture extension for encoding local and global obscurant and night related behaviors is developed. This extension is not only efficient enough to handle dynamically-changing smoke effects, but also generic enough to handle other environmental effects such as fog, rain, night, etc. Using the architecture, the LPB group eliminates the above mentioned behavioral limitations and transforms the existing ModSAF behaviors to phenomenology sensitive behaviors.

This paper starts with the brief discussion of the current ModSAF phenomenology. It is followed by approaches taken for phenomenology behavioral implementation. This includes the phenomenology behavior architecture, behavioral classification concept, and actual phenomenology behavioral implementation.

3. Current ModSAF Phenomenology

The DVW (Dynamic Virtual Worlds) group of LADS has introduced environmental effects within ModSAF. Their contributions include an environmental architecture and associated environmental models (Anon, 1995).

3.1 Environmental Architecture

The environmental architecture in ModSAF was designed to allow software modules that simulate environmental phenomena to easily be plugged into ModSAF. A well-defined API isolates the implementation details of the environmental models from the user. This approach is advantageous in that it allows models to be switched in and out easily, facilitating multi-resolution modeling, and making

ModSAF more adaptable to different applications requiring different models.

The API is implemented in a library called libenvironment. Each environmental model is contained in its own library. Each model registers its capabilities with libenvironment. Some models provide information and can only be queried, while other models can carry out requests for man-made environmental effects such as smoke clouds and illumination flares.

When the API is asked for a piece of information about the environment, such as a query for temperature at a given location or a query for transmissivity between two locations, it dispatches the request to the model(s) that know how to handle it. If several models know how to handle a given request, a resolver model is used to merge the information into one response. This way, several models which provide the same type of information can work together. For example, consider model A, a simple, constant ambient model, and model B, a model that simulates the smoke effects of a burning tank. Both transmissivity values of the ambient and the smoke of the burning tank will contribute to the transmissivity of near the tank. If the user makes a query for the transmissivity of near the tank, it is up to the resolver model to calculate the resultant transmissivity based on the inputs of the two models. Specifically, the resolver model multiplies the two transmissivity values from the two model to compute the resultant transmissivity. The way of merging values from multiple models varies depending on the nature of a parameter (roughly equivalent to environmental effect) activated by a query.

3.2 Environmental Phenomena Currently Modeled in ModSAF

This section describes the environmental phenomena which are part of the 2.1 version of ModSAF (Anon, 1995).

3.2.1 Constant Environment

There is a simple constant environmental model which is capable of providing answers to all possible queries supported by libenvironment. Running ModSAF with this model simulates running an exercise on a clear sunny day at high noon. This basically simulates SIMNET environment. However, it can be used for DIS if computationally expensive dynamic environment is not required for a specific DIS exercise. Or some environmental factors (called parameters) can be selectively registered to this constant model, but others are assigned to more complex environmental model. For example, the

constant model provides constant temperature, wind, and transmissivity, but a smoke model can provide detailed transmissivity.

3.2.2 Simple Environment

A collection of simple, low-fidelity models constitutes the simple environment model. Although this model supports time varying parameter values, most of parameters remain constant throughout the exercise. Those parameters are individually initialized. Once they are set, all parameter values are assumed horizontally and vertically constant. This model always checks legality of those values; such as range check and cross validity check with related values. Again, parameters are individually registerable to any environmental model.

3.2.3 Natural Illumination

The natural illumination model takes the sun position, moon position, and moon phase into consideration when calculating illumination. This model determines the value for illumination. This is based on the U.S. Army Research Laboratory Battlefield Environment Directorate's (ARL-BED) ILUMA model, as driven by commonly available model of the ephemeris.

3.2.4 Smoke and Dust Clouds

The COMBIC (Combined Obscuration Model for Battlefield-Induced Contaminants) model is used to calculate transmissivity through smoke and dust clouds. It was developed by the U.S. Army Research Laboratory Battlefield Environment Directorate, and consists of the following:

- smoke sources description
- diffusion model (simulates the expansion of the cloud)
- buoyant rise model (simulates the rising of warm clouds)
- boundary layer model (considers wind, temperature, and cloud density)

The COMBIC model is used both in preprocessing and at run-time. During preprocessing, the model is used to produce data for specific munitions types and environmental conditions. Smoke-history tables are also computed off-line. During run-time, once the actual ammunition type and environmental conditions are known, the actual smoke characteristics information is obtained from the tables. This information is then used to calculate the transmissivity.

3.2.5 Atmospheric Extinction

The LOWTRN model is used to compute uniform atmospheric extinction (or transmission). The output from LOWTRN is a table that provides transmissivity at various ranges given a particular set of input atmospheric conditions. Tables of extinction coefficients for useful sets of atmospheric conditions are precomputed and accessed at run time, since the LOWTRN model is too slow to run in real time. The LOWTRN model is developed by the U.S. Air Force Geophysics Lab.

3.2.6 Signal Smoke and Flares

Signal smoke and flares are intentionally-generated man-made environmental phenomena used for signaling. An enabling task that monitors for these signals can be used to enable the signals to act as conditions for units transitioning from one phase of the execution matrix to the next.

4. Behaviors

Behaviors are entities capable of producing interactions between systems (Kwak, 1995). Behaviors also interact with each other at the system boundaries, and these interactions are often considered as interactions between systems. Therefore, behaviors define observable characteristics and capabilities of a system. However, behaviors cannot exist by themselves. They need embodiment. A system is such an embodiment or a container of the behaviors.

Two or more behaviors can be composed to create a composite (or complex) behavior, and this compositional relationship can be recursive. Therefore, depending on our point (or level) of interest, some behavior can be viewed as a composite behavior, while the same behavior can be seen as a primitive (or atomic) behavior for other behaviors.

A composite behavior is not a simple collection of primitive behaviors. When the primitive behaviors are aggregated for a composite behavior, extra behavioral characteristics are added. The extra could be additional memories (or states) and state transition logic¹. Obviously, the way of aggregation using operators, such as AND, OR, etc., greatly influences the characteristics of the composed behavior.

For our research and development of Phenomenology behaviors in ModSAF, unit behaviors (or platoon behaviors) are treated as composite behaviors.

¹States and state transition logic approach is a finite state machine based behavioral implementation. Many other approaches are available (Kwak, 1995).

Individual vehicular behaviors are treated as primitive behaviors for the unit behaviors. Because the unit behaviors have directly observable and controllable from the outside of ModSAF, they should be meaningful and useful as they are to the external world (or a human operator). That is, they should be able to perform desired, identifiable, and meaningful tasks; such as road march, occupy position, etc. Therefore, when such a task is assigned to a tank platoon, using internal behavior logic and memories it controls vehicles in the platoon through activating vehicle-level behaviors; such as a vehicle movement behavior. This type of a behavioral organization constitutes a behavioral hierarchy in ModSAF.

The ModSAF behavioral hierarchy is mapped into four classes of behaviors: i.e., *Reflective* (or *Active*) behaviors, *Reactive* behaviors, *Reflexive* behaviors, and *Passive* behaviors. The reflective behaviors correspond to the composite behaviors described above. The reflexive behaviors are primitive (or atomic) behaviors for the unit behaviors. The reactive behaviors are unit-level exception handlers that are activated when the unit confronts an unexpected situation. They also temporarily suspend the unit behavior currently executed when they are activated. If such situation is not persistent any longer, then the reactive behavior becomes inactive and the suspended unit behavior becomes active again. The passive behaviors are encapsulated in ModSAF physical components. They are not regular ModSAF behaviors but needed to simulate behaviors of physical components.

In fact, categorization of behaviors into the four classes is based on time planning horizon, degree of deliberation, and degree of coordination. Obviously, reflective behaviors, such as ModSAF unit level behaviors need much higher degree of planning. Thus, they are associated with long time planning horizon, high degree of deliberation, and high degree of coordination between vehicles, while characteristics of reflexive behaviors such as ModSAF vehicle level behaviors are represented with shorter planning horizon, little deliberation, and no coordination crossing the vehicle boundary.

Four categories of phenomenology behaviors classification with descriptions are listed as following:

- **Passive Behaviors**

Performance of tasks not directly related to environmental effects is either degraded (in most cases) or improved. These are related to sensors and hulls (ModSAF physical libraries), and degradation of weapon-system

effectiveness (ModSAF weapon delivery libraries).

- Reflexive Behaviors

Default vehicle-level reflexive behavior. No coordination with other vehicles is needed or required. Slowdowns and heading changes are examples.

- Reactive Behaviors

Unit-level behavior which is initiated by any unexpected environmental change. Loosely speaking, this is a unit-level reactive phenomenology behavior. It includes units moving to alternative positions, changing the direction of travel, unit formation change, or employing protective smoke under a sudden enemy attack.

- Reflective (or Active) Behaviors

Unit-level behavior which coordinates with deliberation. Tactical maneuvering to take advantage of deliberately-deployed smoke can be an example.

The above four classes of behaviors are inter-related each other, and in most of cases they run concurrently. (Reactive behaviors are only exception. They are usually dormant until a proper condition is arrived.) They are usually cooperative for a common objective that is given to a reflective behavior for execution. However, a conflict can be arisen: sometimes planned behaviors from a unit cannot be executed by the subordinate individual vehicles because of constraints imposed on individual vehicles. Even though a unit plans out a unit behavior with consideration of the constraints of its subordinates, it is only able to consider the average environmental effect. Each vehicle might confront a totally different environmental condition from the averaged condition used in the planning. Thus an unit level phenomenology behavior, such as smoke sensitive phenomenology behavior, cannot satisfy all constraints of the individual vehicles in the unit. The requested unit behavior should be reevaluated by the individual vehicles before executing it. In the worst case, the planned unit behavior can threaten the safety of an individual vehicle. This type of localized behavioral reevaluation and modification is expected to be performed by reflexive behaviors at the vehicle level. Therefore, the reflexive behaviors have the highest priority among them. (Passive behaviors are always executing to simulate physical components. Thus, they are not included for discussing behavioral priorities.) The second highest priority is given to reactive behaviors. They are allowed to interrupt the unit behaviors (reflective behaviors) assigned to the

unit in order to handle unexpected situation. Thus, the reflective behaviors have the lowest priority.

The highest priority of reflexive behaviors allows individualistic behavioral modification by deviating from a command given from an unit level behavior. For example, if smoke blocks the movement of only one vehicle in an unit, then the blocked vehicle should adjust behavior to overcome the newly confronted environmental condition. It might change speed and/or heading depending on the location and density of smoke blocking its movement while others are moving as commanded. Thus, the blocked vehicle acts differently from the rest of the vehicles. This uncoordinated individualistic vehicle behavior with respect to other vehicles in the unit will increase disorder in the unit. Thus, the group behavior of the unit will be automatically degraded. This type of behavioral disturbance influencing at the unit level implements an *emergent (unit) behavior* that is not explicitly programmed but surfaces up from the interactions of many entities that are capable of responding individually to local environment².

5. Behavioral Scaling

Under phenomenological effect, a behavior of a vehicle or a group of them tends to perform poorly. For example, the density of smoke is increased, the behavioral performance is degraded. That is, thicker smoke makes a vehicle move slower. If the vehicle is relieved from the smoke, then it has to restore the original traveling speed. A general relationship between environment and perception and environmentally sensitive behavior is shown in Figure 1. Because of standard terminology of the military perception levels, finer description is made for the perception performance as the smoke density is thicker. However, the behavioral performance scale is very coarsely described because of lack of such terminology. In either case, there is no crisp boundary in such behavior performance descriptions. Even though this discussion is made based on smoke, this concept can be applicable to any environmental effect that reduces perception performance. Fog, night, rain, snow, dust, and etc. are such environmental effects (Kwak et. at. 1995).

²Pure subsumption architecture is solely based on emergent behaviors (Brooks, 1986), which is an optimistic behavioral implementation. Thus, there is no guarantee to achieve a global objective. In this paper, usage of emergent behaviors are limited to perturbation of a globally coordinated behavior to properly respond environmental irregularities so that the unit (or group) can still be able to achieve the given global objective.

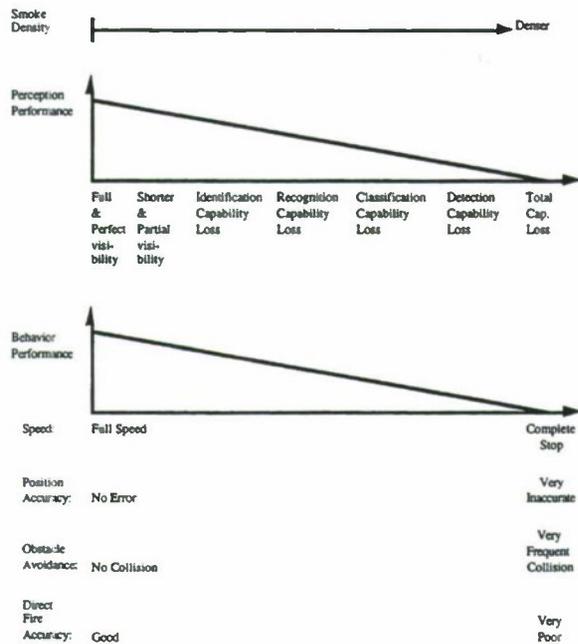


Figure 1: Perception and Behavior Performance under Smoke Environment

One of the noble approach developed for this project is *behavioral scaling*. In stead of creating discrete, multiple, and specialized behaviors for various cases of environmental effects, a single behavior based on assumption of a high noon clear day is created, which is called a *base behavior*. Then the base behavior is dynamically modified (or specialized) to match with the given environment. Thus, a wide (or whole) degree of a given environmental effect can be covered by one base behavior by scaling the behavior. This approach eliminates an enumeration of multiple behaviors, which are similar but vary depending on a degree of the given environmental effect. Economy of behavioral implementation is obviously a big advantage of this approach.

However, significance of this approach comes from a paradigm match between the behavioral implementation and human behavior. We, human beings, continuously monitor environment surrounding us and dynamically adjust our behaviors based on perceived information about the world. If we cannot perceive anything because of a very poor environmental condition, then we have to stop a currently executing a *task* - a goal directed behavior - for our safety (or self-survival). If we are able to start to perceive the world again, we can resume our task within the limit that the environmental condition permits. For example, obscuration and/or lower light level limits our perception capability. Thus, under such an environmental condition we move slowly with caution. If such a condition is removed, then we

can move freely as we intended. This type of dynamic behavioral adjustment based on perception (or in general environmental assessment) is behavioral scaling. In ModSAF, like ourselves, phenomenology behaviors are scaled based on an environmental assessment level (or perception level). This adjustment continues until the behavioral performance is scaled to a proper level.

Finally, the behavioral scaling approach provides a big relief for knowledge acquisition process for phenomenology behaviors. First, quantitative behavioral descriptions related to a certain phenomenology are very scarce. Even through some of them are available, the description is hardly complete. For example, a formation interval for a tank platoon is described as following: "During clear day, the formation interval is 100 meters, but if visibility is limited, the interval should be half of the clear day interval." This is one of the best descriptions we can get from SME input. However, this statement already has two issues to be implemented in ModSAF. First, how much degree of visibility degradation really means "limited". Second, even though we can know the exact degree of the degradation, the description covers only one case. There are whole spectrum of visibility degradation. Therefore, the captured knowledge conveys very minimal information. If this statement were literally adopted for writing a behavior in ModSAF, then the spacing adjustment would be binary; for example either 100 meters or 50 meters intervals, not in between. This would be hardly realistic neither. If non-binary behavioral adjustment is needed, then obviously many entries for formation spacing depending on varying degree of visibility degradation are required. This is practically impossible because of lack of availability of such amount of data. Therefore, behavioral scaling is seemingly only option to implement realistic phenomenology sensitive behaviors. However, the captured knowledge by SME's is still useful. The knowledge, instead, was used to verify the behavioral scaling scheme by making observation of the implemented behavioral output; i.e., measuring intervals under a typical degree of an environment effect (for example, by a medium level of fog density) during development. If the observed result was not satisfactory, the behavioral scaling scheme was tuned. This process continued until the result was satisfactory. SME's was involved in this process.

Consequently, the behavioral scaling approach is an innovative methodology. It allowed to create realistic phenomenology behaviors in ModSAF by overcoming lack or vagueness of data. This approach provides following advantages: implementation economy with dynamic specialization, high realism due to paradigm

match with that of human perception-based behaviors, and a seemingly only practical means of implementing efficient and realistic phenomenology behaviors.

6. Phenomenology Behavior Architecture

Introducing phenomenology behaviors into ModSAF was first seen as a formidable task. It was initially estimated that all existing ModSAF behaviors should have been examined and possibly modified to implement the phenomenology behaviors case-by-case. Because of the large number of ModSAF behavior libraries - far more than 100, such a task would have required significant amount of engineering efforts. However, we developed an approach for the implementation of the phenomenology behaviors which is complementary to the existing ModSAF behaviors and maintains the hierarchical behavioral structure.

As discussed before, the current ModSAF implement behaviors as following: physical component libraries implement passive behaviors. Vehicle level task libraries realize reflexive behaviors. Reactive task libraries construct reactive behaviors, and finally unit level task libraries implement reflective (active) behaviors (Kwak et. al. 1995). This behavioral organizational structure exactly matches with the phenomenology behavior implementation architecture developed and shown in Figure 2. This architectural match has significantly reduced the development time required to incorporate the phenomenology behaviors into ModSAF.

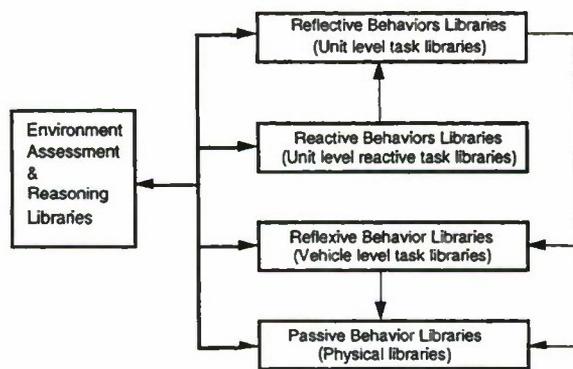


Figure 2. Interactions between environmental libraries and other libraries.

6.1 Environmental Assessment and Reasoning

ModSAF phenomenology behaviors are built upon two core libraries; i.e., libenvassess and libenvreason. Whenever there is an environmental change, the

ModSAF behavior libraries can consult environmental impacts on ModSAF behaviors with the assessment and reasoning libraries. Then the libraries return behavioral modification information to the ModSAF behavior libraries. Based on the information, the ModSAF libraries adjust their behaviors. The relationships of two libraries are shown in Figure 3.

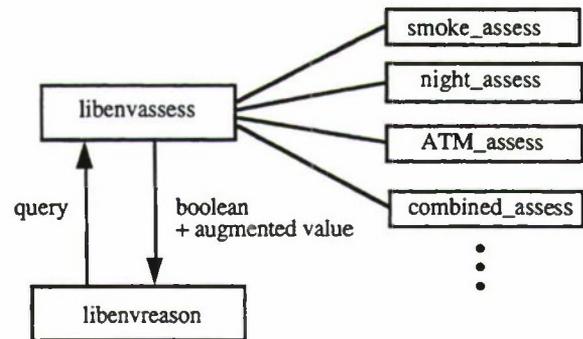


Figure 3: Environmental Assessment and Reasoning Architecture

Libenvassess is an environment perception library. When there is a query about the environment from other libraries, it utilizes libenvironment and all available sensor libraries if needed. Then, it processes numeric values received from such libraries and returns a symbolic value in response to a query as shown in Figure 3. It usually answers a query with a Boolean value and augmented values if applicable. Thus, it can be considered as an abstract symbolic sensor. In other words, libenvassess is an interface library translating numerically measured sensory information to symbolic values for behavioral libraries. Because of the modular approach, each environment factor can be added piece by piece when available. Currently, three different classes of environmental effects are considered. They are smoke, night, and uniform atmospheric effects. These effects are independently computed in ModSAF. That is, smoke is represented as obscuration, night is as a low illumination level, and uniform atmospheric effects are as poor transmissivity. Not all behaviors are expected to be simultaneously influenced by all of these environmental effects. Some might be susceptible to only one specific environmental effect, while others might be influenced by all of them. Therefore, libenvassess currently concerns four different cases; i.e., smoke only effect, night only effect, atmospheric only effect, and finally combination of all three effects³.

³The current architecture is able to support any combination of smoke, night, and atmospheric

Specifically, those individual effects are separately computed by assessing obscuration, illumination, and transmissivity values, which are retrievable through libenvironment's public interfaces and processed individually. However, in order to combine all three effects, spatial frequency (SF) approach is chosen (Mackey et. al. 1992). That is, for the given sensor, libenvassess computes SF based on apparent contrast and illumination level. The block diagram for SF computation is shown in Figure 4.

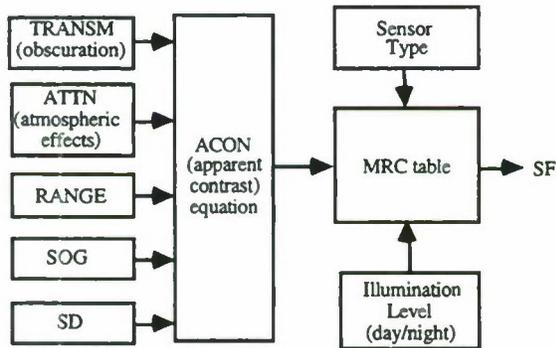


Figure 4: SF computation block diagram

In order to compute SF, apparent contrast(ACON) is calculated based on five different aspects. First, obscuration effect(TRANSM) is calculated. This value is affected by existence of smoke and dust. If smoke or dust is presented transmissivity is degraded. Second, uniform atmospheric effect or extinction coefficient(ATTN) is calculated based on current environmental conditions given from libenvironment. This value is affected by fog, rain, snow, etc. Third, range(RANGE) is the sensor to target distance in kilometers. Fourth, it gets sky_over_ground ratio(SOG) from libenvironment. Fifth, suppression_degradation(SD) is currently set to 1.0.

After these five values are computed, the following equations are used to calculate apparent contrast(ACON):

DVO, NVO:

$$ACON = (SD * TRANSM) / (1.0 + (SOG * (\exp(ATTN * RANGE) - 1.0)))$$

IR:

$$ACON = SD * TRANSM * (\exp(-1.0 * ATTN * RANGE))$$

where

DVO: Daytime View Optics
 NVO: Nighttime View Optics

effects. However, not all of external interfaces are currently provided.

IR: Infrared or Thermal Sensor.

Illumination level is computed by ILUMA model that takes the sun position, moon position, and moon phase into consideration.

When the illumination level is computed, it allows to pick a MRC (Minimum Resolvable Contrast) curve as shown in Figure 5. Then simply read Y axis value corresponding to the apparent contrast represented in X axis.

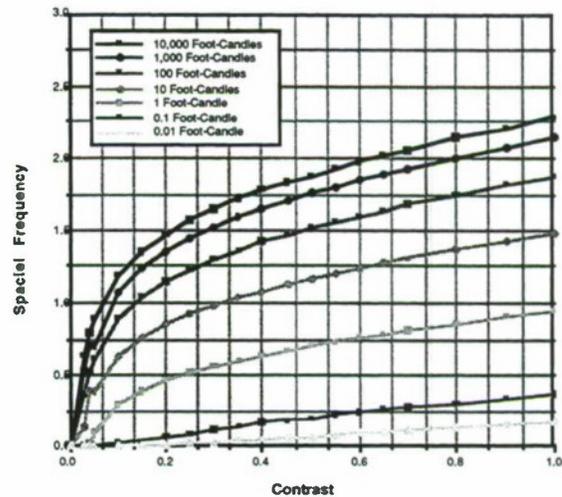


Figure 5: MRC Curve

After SF is computed, it is compared with a prescribed value. If it is greater than the prescribed value, then there is no behavioral degradation. If it is smaller, then there is a behavioral degradation. The degree of the degradation is computed based on the magnitude of SF. Our implementation for computing the degree is to normalize SF - a behavioral scaling value, so that it can have a range of zero to one. That is, if the environment becomes very unfavorable, such as a dark night-time dense fog, then the normalized value becomes very close to zero. However, the value becomes one under a high noon clear day. If the environment is in between the two extremes, the value lies anywhere between zero and one. Details can be found in ModSAF libenvassess Texinfo which is distributed with ModSAF source code distribution.

Libenvreason is a reasoning library for environment. It mainly makes queries to libenvassess and performs reasoning functions to take environmental effects into account. The outcome of this reasoning is a behavioral scaling value to be used to modify behaviors under consideration. Most of ModSAF behavior libraries communicate with this library to get the behavioral scaling values.

6.2 Phenomenology Behavior Implementation

Passive and reflexive behaviors, which are fundamental behaviors, are strongly influenced by phenomenology effects because they are very closely tied to physical models and physical phenomena. These libraries interface environment through libenvassess and libenvreason. Thus, the libenvassess and libenvreason form the basis upon which all passive and reflexive behaviors have been developed. In turn, the passive and reflexive behavior libraries in ModSAF provide a foundation for other behaviors, such as reactive and reflective behaviors. That is, many reactive and reflective ModSAF behavior libraries use the passive and reflexive behaviors to complete their behaviors.

In order to facilitate further discussion, behavior libraries in ModSAF are grouped into two:

- Passive and Reflexive Behavior libraries, and
- Reactive and Reflective Behavior libraries.

Each of the libraries of the latter group, especially the reflective behavior libraries, are individually created for implementing specific tasks. The libraries in the former group are generic libraries that implement fundamental behaviors, such as sensing, shooting, and simple movements. These libraries are not fully specialized for performing any specific and stand alone tasks. Instead they provide atomic behaviors for the latter group so that the construction of the specifically tailored reactive and reflective behaviors can be facilitated. This generality and commonality reduces the number of separate libraries which is required for passive and reflexive behaviors.

The common environmental assessment and reasoning libraries, the small number of passive and reflexive behavior libraries, and the utilization of these libraries in the construction of reactive and reflective behaviors have significantly reduced the size of the required development efforts. First, the approach developed for the phenomenology project encourages the solution of generic problems rather than specialized problems. For example, the concept of behavioral scaling was used. Rather than tailoring behaviors on a case-by-case basis, the existing behaviors⁴ were scaled to transform them to environmentally sensitive behaviors. Thus, the behavioral scaling was effectively used for a means of specialization. Therefore, by modifying the small number of generic passive and reflexive behavior

⁴The existing behaviors become the base behaviors that phenomenology sensitive behaviors can be derived from.

libraries, the environmental sensitivity of these libraries became immediately effective to the large number of behaviors represented by the reactive and reflective behavior libraries too.

Moreover, the generality of environmental assessment and reasoning can handle many distinct classes of environmental conditions which are being developed by LADS in DARPA's Synthetic Environments initiative (Dynamic Virtual Worlds and Dynamic Terrain & Objects). Thus, when a new environmental effect was introduced, the existing assessment and reasoning could properly function with virtually no modification. If not, a very minimal upgrade process was needed to provide new interfaces to the libraries to handle a newly required functionality.

In the phenomenology behaviors project, we chose to use generic interfaces provided by the environmental assessment and reasoning libraries for most of cases. Thus, the passive and reflexive libraries that use these environmental interfaces were generic enough to be equally applicable to the three currently implemented classes of environmental conditions: smoke, night, and uniform atmospheric conditions.

The reactive and reflective behavior libraries have structured relationships - not totally hierarchical, but logically organized. This characteristic facilitates the inclusion of phenomenology sensitive behaviors. That is, if one common library is modified, then many libraries automatically take advantage of this modification. However, these libraries have been individually examined, unless excluded because of a total detachment from phenomenology. If a library was determined to require modification, it was modified to call common libraries that had been modified to be environmental sensitive. This type of modification is recursive, and usually terminated at the most complex behavior libraries that are usually directly used by human ModSAF operators through ModSAF GUI or by CCSIL commands.

7. Current Status of Phenomenology Behaviors

In ModSAF currently three categories of behaviors - sensing, movement, and shooting behaviors - properly respond to under various phenomenology effects; such as smoke, day/night, and fog, etc. These behaviors are fundamental to any combatant in a battle field. If there is an environmental effect, then performance of sensing, movement, and shooting behaviors are degraded. On top of these fundamental behaviors, two smoke specific behaviors; smoke reaction and smoke deployment are added to ModSAF. When one of these reactive behaviors executes, the

above fundamental behaviors also run background while matching up their performance level to a given environment condition. Currently, all of the phenomenology behaviors are mainly optimized for both Army ground vehicles and Army combat echelons; such as a M1 tank platoon.

7.1. Sensing

The primary sensor of a ModSAF ground vehicle is a vision sensor. Thus, enemy vehicle information is collected mainly through the vision sensor. Thus, if vision sensor's performance is degraded by an environmental effect, then ModSAF vehicle's ability to acquire a target is automatically degraded. The performance degradation is based on the upper mentioned SF. A ModSAF vehicle has an ability to incorporate multiple crew member sights in order to simulate multiple human crew members. Each crew member's sight also has multiple sensors with different types, but only one sensor is allowed to be used at a time by each crew member in ModSAF. ModSAF also automatically chooses the best performing sensor among the given sensors to each crew member to match with an environmental condition.

When a specific sensor is chosen, a target spatial frequency is calculated and compared with the minimum cycles required for detection, classification, recognition, and identification - four target acquisition levels in order. For example, if the spatial frequency is higher than the minimum detection cycle and lower than the minimum classification cycle, then the target is classified as a detected target. That is, a smaller SF results in a lower target acquisition level.

Target acquisition process in ModSAF is a cooperative task among the crew members in a vehicle. If enemy targets are spotted, then the enemy target information observed from the multiple crew members are merged into one unified enemy list. Then the list is processed to assign threat levels to the targets, and a target with the highest threat level is chosen for an engagement.

7.2. Movement

ModSAF vehicle movement is influenced by environmental effects. For example, if there is fog, then our common sense reasoning tells to slow down traveling speed, and it continuously adjusts traveling speed until the traveling speed is slow enough for our safety under the environmental condition. The degree of slowness is also roughly proportional to the density of smoke. That is, the denser fog is, the slower the traveling speed is. In ModSAF, the denser fog in ModSAF leads to a lower spatial frequency,

and then the lower frequency is translated into a behavioral scaling value (or a behavioral degradation value). Finally, it causes to slow down the vehicle speed.

Whenever there exist environmental effects, the movement behavior does not always need to be degraded. If a commanded speed is slow enough so that a vehicle can maintain the speed, no speed reduction is applied. That is, no behavioral degradation value is generated. This feature is built in ModSAF libenvassess and libenvreason. Because of the feature, if a proper speed is given to a ModSAF vehicle, then there is no speed reduction even under an environmental effect. However, if an excessive speed is commanded, then a ModSAF vehicle refuses to follow the commanded speed. The speed is automatically adjusted to a proper speed. Therefore, no matter whatever speed is commanded, the ModSAF phenomenology movement behavior maintains a proper speed corresponding to the given environment.

Currently, ModSAF environmental effects are globally uniform except the smoke effect. Thus, while most of environmental effects only cause speed reduction, the smoke effect, which is localized, opens up another option to respond - a heading change. Rather than going through the smoke with a slower speed, a ModSAF vehicle is allowed to go around the smoke cloud. Thus, it can deviate from the planned path. Because deviating from the given path introduces a greater surprise to the plan/commander of the ModSAF vehicle than a simple speed reduction, ModSAF has a GUI-based switch to enable or disable the heading change behavioral option during execution. When this switch is disabled, a ModSAF vehicle simply responds smoke with a slower speed. If this switch is enabled, then a ModSAF vehicle tries to avoid the smoke by modifying its heading. Sometimes, a ModSAF vehicle briefly travels through a smoke cloud while it tries to avoid the smoke cloud by following the perimeter of the cloud. Because of the dynamic nature of smoke cloud formation, this situation cannot be totally avoidable. However, if a ModSAF vehicle is caught in a smoke cloud while avoiding, it automatically reduces its traveling speed. Thus, two behaviors are superimposed.

Movement behavior is only affected by a driver's sensing ability unlike the cooperative target acquisition behavior described above. Therefore, driver's sensor ability is the sole source for determining the movement performance. If a driver has no night vision device, such as a night vision goggle, then the driving performance at night will be significantly poor; i.e., moving very slowly. If a

driver has a night vision device, then its movement is greatly improved but cannot match with the daytime performance.

7.3. Shooting

Shooting behavior of ModSAF vehicles is affected by environmental effects, too. Specifically, two sub-behaviors of the ModSAF shooting behavior are influenced. First, delivery accuracy of a direct weapon is affected. If there is an environmental effect; i.e., other than a high noon clear day, then the delivery accuracy is degraded. The degree of degradation is again computed based on the spatial frequency. That is, the low spatial frequency is translated into a behavioral degradation value, and this value is used to magnify a shooting error - dispersions and biases (Topper, 1993). This scheme simulates less accurate delivery performance of a direct firing weapon. The behavioral degradation value, as discussed before, is computed by libenvreason using the assessment from libenvassess. Second, targeting behavior is also affected by environmental effects. Targeting difficulty is simulated with a longer time required to track a target before shooting with a direct firing weapon. The behavioral degradation factor is again used to lengthen the tracking time longer.

Performance of the above shooting sub-behaviors is only determined by gunner's sensing ability in ModSAF. For example, an US tank gunner has a thermal sensor. Thus, the gunner has little difficulties to look out to aim and shoot an enemy target at night or through a smoke cloud if the smoke is not IR blocking smoke. However, if a gunner had no thermal sensor, then his ability to engage a target would be significantly degraded under such environment.

7.4 Smoke Reaction Behavior

This is a specialized behavior for an US Army tank platoon. When a smoke screen blocks an intended traveling path, a tank platoon changes its formation to a line formation and reduces its formation spacing distance. This smoke reaction behavior is activated only when this reaction behavior is enabled. This option can be switched on or off through ModSAF GUI. However, the smoke reaction behavior is coupled with the smoke avoidance behavior option. That is, if the smoke avoidance option is enabled, then the platoon will go around smoke rather than going through the smoke. The smoke reaction behavior is effective when the smoke avoidance option is disabled. If neither options are enabled, then the platoon will go through the smoke screen without platoon level coordination. They will neither go around the smoke nor switch the formation

with a reduced formation spacing distance between vehicles.

When the platoon emerges from the smoke screen, it starts to travel in a dash mode - a faster speed with a fully alert state. If any enemy is encountered, then the platoon immediately attacks the sighted enemy with the tank main guns. Then it will continuously attack the spotted enemy until the enemy is destroyed. If no enemy is spotted after emerging from the smoke screen, then it will keep traveling in the dash mode until a prescribed time limit is met without encountering any enemy vehicle. If so, the smoke reaction task is terminated and the platoon starts to execute a suspended task due to the smoke reaction behavior.

7.5 Smoke Deploy Behavior

This is also a specialized behavior for an US Army tank platoon that carries on-board smoke grenades. If any one of the tanks in a platoon is attacked by an enemy anti-tank guided missile (ATGM), then the tanks of the platoon start to deploy its smoke grenades to hide themselves from further observation from the enemy ATGM gunner, and shoot HE rounds back to the enemy ATGM site while covering or concealing them against the enemy. If the enemy ATGM site is completely destroyed from this engagement, then the platoon resumes its temporarily suspended task due to the enemy ATGM attack.

This behavior actively deploys smoke between the tank platoon and the enemy ATGM site. Thus, accuracy of ammunitions from the tank main guns as well as visually guided missiles exchanged is degraded when they fly through the deployed smoke clouds. The performance of vehicle movement is also degraded by the deployed smoke. Finally, target acquisition and detection of both parties also become less effective. Sensing, movement, and shooting are equally affected by the deployed smoke even when a smoke deploy behavior is executed.

7.6 Summary of Available Behaviors by Categories

The following list provides a synopsis of the phenomenology behaviors capabilities implemented:

- Environmental assessment and reasoning:
 - Libenvassess: assessment of environmental effect
 - Libenreason: computation of behavioral degradation value based on environment assessment
- Complete set of passive and reflexive behaviors for all types of Army ground vehicles:

- Passive sensing: target acquisition
- Passive shooting: weapon delivery accuracy
- Reflexive sensing: automatic sensor switching
- Reflexive shooting: tracking
- Reflexive movement: speed reduction, heading change, active formation keeping
- Two reactive tasks:
 - Libureactsmoke: smoke cloud crossing with formation and speed changes
 - Libusmoke: smoke deployment against enemy anti-tank missile attack
 - Visibility based dynamic formation spacing adjustment
- Movement related reflective tasks for Army ground vehicles
 - Phenomenology influenced hiding positions for hasty occupy position

8. Summary and Conclusions

The phenomenology behavioral architecture with libenvassess and libenvreason provides a generic means to interface to ModSAF phenomenology. This approach facilitates inclusion of phenomenology behaviors in ModSAF. Rather than creating specially tailored ModSAF behavior libraries, the existing behaviors are transformed to phenomenology-sensitive behaviors by simply scaling the behaviors using values from libenvassess and libenvreason. This novel approach was found to be applicable to a wide range of ModSAF behaviors.

Not all behaviors are scalable. Some behaviors are very specific and only valid to a certain environmental condition. For example, reaction to smoke screen is not a scaled behavior from a general movement behavior. This behavior must be specifically created.

Sensing, movement, shooting behaviors for Army ground vehicles in ModSAF are successfully implemented with the behavioral scaling concept. However, development of the phenomenology behavioral architecture is still under way. As the project continues, the architecture will progressively cover more behavioral concepts. However, phenomenology behavior project is already mature enough to find applications to other services, such as Marine Corps, Navy, and Air Force utilizing the generic means of phenomenology behavior architecture, and the behavioral scaling concept. The success on Army phenomenology behaviors can be easily duplicable for other services.

Finally, all of the discussion done in this document is based on ModSAF 2.1.1. Libenvassess and

libenvreason as well as all of environment sensitive libraries discussed can be found in ModSAF 2.1.1.

9. Acknowledgment

This work is being supported by the USA Army STRICOM ADST program under contract number N61339-91-D-0001.

10. References

- Anon (1995). *Environmental Stealth/ModSAF User's Guide*, Lockheed Martin, Cambridge, Masss, Nov, 1995.
- Anon (1996). *ModSAF Software Architecture Design and Overview Document (SADOD)*, Lockheed Martin, Cambridge, Mass, Jan, 1996.
- Brooks, R. (1986). "A Robust Layered Control System for a Mobile Robot", *IEEE Journal of Robotics and Automation*, Vol. RA-2, No. 1, pp. 14-23.
- Kwak, S. H. (1995). "A Comparison Study of Behavioral Representation Alternatives", *Proc. of the Fifth Conference on Computer Generated Forces and Behavioral Representation*, University of Central Florida, Orlando, Florida, May 9-11, 1995.
- Kwak, S., Longtin, M., Patel, B. (1995). *Initial Study on an Implementation of Phenomenology Behaviors in ModSAF*, Technical Report, Loral, Cambridge, Mass, Aug, 1995.
- Mackey, D, Dixon, D., Jensen, K., Loncarich, T., Swaim, J. (1992). *CASTFOREM (Combined Arms and Support Task Force Evaluation Model) Update: Methodologies*, TRAC-WSRM-TD-92-011, US Army TRADOC Analysis Command-White Sands Missile Range, April, 1992.
- Topper, P. (1993). *A Compendium of Close Combat Tactical Trainer Data Structures, Algorithms, and Generic System Mappings*, US Army Materiel Systems Analysis Activity, May 1993.

11. Authors' Biographies

Se-Hung Kwak is Principal Software Engineer at Lockheed Martin Federal System Advanced Distributed Simulation, where he has led development and application of software architecture and behavioral representation for Computer Generated Forces since 1993. Currently, he is leading the phenomenology behaviors project for ModSAF. He received his Ph.D. and M.S. from Ohio State University in Electrical Engineering in 1986 and 1984 respectively, and his B.S. from Seoul National University in Electronics Engineering in 1979. Before joining

Lockheed Martin, he was Research Associate Professor of Computer Science at Naval Postgraduate School. His research has focused on intelligent robot control and control architecture. He is the originator of the Rational Behavior Model(RBM). His research interests also include Artificial Intelligence, Robotics, Mission Planning Expert Systems, Object-Oriented Distributed Systems, Software Engineering, and GPS/INS Navigation Systems.

MAJ Reba M. Lyons is currently assigned as the ModSAF Project Director at the U. S. Army Simulation and Instrumentation Command, (STRICOM). She received a BS in Special Education from the University of Tennessee at Knoxville and an MGA in Management Information Systems from the University of Maryland. Major Lyons is responsible for ModSAF Program Management and baseline version releases.

Detection Models for Computer Generated Individual Combatants

Douglas A. Reece and Ralph Wirthlin
Institute for Simulation and Training
3280 Progress Dr., Orlando, FL 32826
dreece@ist.ucf.edu

1. Abstract

Computer Generated Forces (CGF) systems that simulate individual combatants require a much more sophisticated detection model than do systems that simulate platforms. Platform-level CGF systems can simplify detection for several reasons: there are often several people on the platform whose observations can be aggregated; the observers may have a limited view of the world through windows or optical devices; visual search may be less important than other searches using instruments; and the platform may make enough noise so that audio detection is irrelevant. These factors are not true for individual combatants, so simulators must model the visual search and audio detection processes in more detail. Furthermore, typical CGF platform entities maintain a simple world model based on entities that they can currently detect; for individual-level simulations in which entities rapidly move in and out of line of sight or field of view, a more sophisticated world model is required. In this paper we describe our visual and audio detection models, an internal world model for individual combatant CGF, and some typical behaviors that result from these models.

2. Introduction

As part of our work in simulating individual combatants (ICs) we have developed a computational model of human detection. The IC simulation is part of our effort to build computer generated forces (CGF) for the Team Tactical Engagement Simulator (TTES). TTES is a training system being developed by the US Marine Corps using distributed interactive simulation technology.

2.1 Detection in platform entities

IC CGF systems require much more sophisticated visual and audio detection models than do systems that simulate platforms. Platform-level CGF systems can simplify visual and audio detection for several reasons:

- There are often several people on the platform whose observations can be aggregated.
- The observers on the platform may have a limited view of the world through windows or optical devices. This situation makes entity facing more important, but eliminates the need to model variations across the field of view of the human observers.
- Visual search may be less important than other searches using instruments. For example, a platform crewman may perform search by looking at a radar display.
- The platform may make enough noise or shield sound so much that audio detection is irrelevant
- The scale of time and space is generally larger in platform level engagements. When targets appear and move they are likely to stay in the same part of the field of view.

2.2 Urban IC detection requirements

At the IC level, the sensing characteristics of an entity must reflect the capabilities and limitations of one human. For visual detection, this means modeling multiple, limited fields of view with different acuities and directing visual search with one field based on objects visible in the second field. Hearing also plays an important part in the soldier's awareness of the situation. Sounds such as weapons fire or footsteps indicate the presence of friendly and enemy troops. Typically, the IC recognizes that other soldiers are around him from movement in his peripheral vision or from sound, and turns his gaze to identify the entity. These multiple sensing modes also require that the IC's mental model of sensed entities be more than just a list of visible objects; the IC must remain aware of threats even as they quickly disappear from his field of view.

Some examples of the interactions with trainees that IC CGF must be able to support include the following:

- Soldier A shifts his gaze from one direction to another. Soldier B is lying prone in the direction A is looking. After some time, A detects B.
- Soldier A shifts his gaze from one direction to another. Soldier B is standing off to the side of A. A does not detect B. B moves, and A detects B. A cannot identify B, but turns toward him and identifies him.
- Soldier A is in a room. Soldier B is outside in the hallway. A and B cannot see one another. A hears B as B walks down the hallway toward A's room.
- Soldier A and B are in a room and hallway as above. A is firing its weapon out the window. A does not hear B walk down the hallway.
- Soldier A is outside facing some direction. Soldier B walks very slowly up behind A. A does not detect B.
- Soldier A moves around the corner of a building and sees soldier B in the street. A moves back behind the corner but prepares to fire at B coming around the corner. When nothing happens, A ventures around the corner to fire at B. B is no longer there. A continues down the street but watches for a nearby threat.

This paper describes the visual and audio detection models, the mental situation model, and some of the related behavior of the IC CGF in TTES.

3. Visual Detection

Ultimately, it would be desirable to model all aspects of human vision, including low level eye movement control, detailed eye characteristics, eye movement attention control, and low level visual processing (feature and motion detection, perceptual memory, etc.). Such detail would provide for accurate simulation of the effects of camouflage, target-background contrast, background clutter, motion detection, light levels, etc. and their effect on combat. However, for now we attempt to capture only some of the vision characteristics and thus some of the tactical effects.

3.1 Vision characteristics modeled

Human vision has been described in various texts such as (Haber, 1980). It is characterized by a high acuity, small angle primary field of view (foveal vision) and a lower acuity, peripheral field of view. The foveal field of view is only about 1°. The fovea is used for shape, pattern, and color discrimination, and

thus is the primary means of identifying targets. We do not model the visual processing in detail but compare the resolution of the foveal vision (a parameter) with the apparent size of an object and probabilistically determine if the object can be identified according to a published formula.

Human vision also has a peripheral area that extends to about 95° from the vision center. Visual acuity falls off rapidly in the peripheral area, dropping to 50% just 1° from center, 15% 8° from center, and finally to zero at the edge of vision. We currently model peripheral vision with a constant acuity value over all angles that is 10% of the foveal acuity. Peripheral vision is sensitive to motion, somewhat less sensitive than the fovea to color, but more sensitive to light. Our model incorporates acuity in comparison to target size and target motion to determine detection probability according to a published equation. We do not consider color contrast, and illumination levels do not currently vary in our synthetic environment.

Since the peripheral field of view is so much bigger than the foveal field of view, it is the primary means of detecting targets. In fact, the near-foveal area is used for identifying all visual features of interest and directing the fovea to new directions. The fovea moves in jumps of 2 - 6° several times a second to new fixation points. We do not model the individual fixations, but instead simulate visual search over an area extending 30° from center (see Figure 1). This is an area in which people can recognize coarse patterns (and direct the foveal vision effectively). Detections

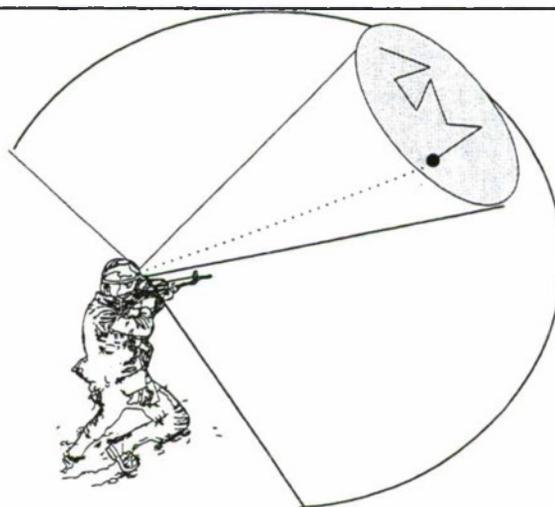


Figure 1. Visual fields of view. Shaded area is primary field of view, in which detailed visual searches are performed.

in the peripheral field of view are effectively immediate; in this 30° area, however, we compute an acquisition time for detecting objects because the fovea has to search the area. This model also assumes that identification is immediate once the fovea fixates on the object.

The above description mentioned a few characteristics of human vision that we have not simulated; others that might be useful in a higher-fidelity simulation include the following:

- Other differences in capability between primary and secondary: color sensitivity, low light sensitivity.
- Effects of low light levels on acuity; effects of high light levels on dynamic range (and thus contrast); adaptation to low or high light levels.
- Effects of observer motion on detection and identification.
- Effect of contrast—brightness, color, and texture. Partly due to computational limitations—sampling background along line of sight to determine its characteristics. A contrast parameter is present in our model, but a default constant is used in all situations. Texture is hard to characterize.
- Effects of clutter in scene on search and detection speed.
- Specific direction of gaze during search and other tasks. However, in the future, search will be limited during specific tasks such as aiming a weapon, moving, or identifying an already detected object, because during these tasks vision is occupied by a specific object in a small angular area and search over the larger primary field of view does not take place.

3.2 Discrete simulation of continuous sensing

There is a modeling problem caused by the discrete nature of our computer simulation. The continuous sensing processes are approximated by discrete sampling. All events in the world and vision system that occurred in the last sampling interval dt are aggregated into the current sample. This sampling causes potential problems with short duration events, with rapid observer orientation change, and with simulation of dynamic processes. First, if the sampling period is longer than duration of just-noticeable events, then these events may be missed. For example, enemy entity peeks head around wall, then pulls it back quickly. Since in our simulations these events are not produced by an easily

recognizable action (unlike weapon firing events), the only solution is to sample frequently. Heuristics may be used to guess when samples can be skipped or the sample period can be stretched (e.g., Rajput, 1995).

Second, if the entity is moving its field of view quickly, e.g. by rotating, then some angles—and thus areas of the world—may be skipped by the (enlarged) foveal field of view. For rapid movements this may be realistic; it corresponds to reduced acuity during observer movement. However, in a slower “sweep,” it is unrealistic to prevent an entity from identifying an object that his vision passed over. We avoid this problem by using a suitably high sampling rate so that the new foveal field overlaps the old one. If this is not possible, then the foveal field is extended to cover the entire swept angle if the angular rate of change is less than 90° per second.

We do not model low-level dynamic vision processes such as light level adaptation, eye movement control, etc. If these were modeled, then sampling must be frequent enough to avoid aliasing or instability.

3.3 Algorithm

The IC sighting algorithm is based on the Army’s CECOM Night Vision and Electronic Systems Directorate (NVESD) sighting model as reported in (Lind, 1995). Versions of this model are used in the Janus and ModSAF simulations. This model has been extended to reflect the visual characteristics described in section 3.1 above.

The output of the algorithm is a sighting status, which is one of the following:

- INVISIBLE - The target is not in the observer’s field of view or the observer doesn’t have a clear line of sight to the target.
- VISIBLE - The target is in the observer’s field of view of sight and the observer has at least a partial line of sight to the target.
- DETECTED - The observer has detected an entity.
- RECOGNIZED - The observer knows the class of the target.
- IDENTIFIED - The observer knows the specific type of the target, and whether it is friend or foe.

The input parameters to the algorithm include the following environmental factors:

- Light level. Since, as mentioned above, we do not consider light level, a default value is used.

- Atmospheric attenuation. We do not currently model this environmental factor. Since the ranges in our application are all under 250 meters anyway, this factor currently has little effect.
- Brightness contrast with background. We do not yet examine the background to determine contrast, so a default value is used here.

The following observer parameters are also inputs to the algorithm. The first two use values from (Lind, 1995); the second two are set to correspond to the description in section 3.1.

- Spatial frequency (acuity) of foveal vision, in cycles per unit angle of the visual field.
- The number of sensor elements (e.g, pixels) that must cover the target in order to have a 50% chance of detecting, recognizing or identifying the target.
- Size of primary and peripheral (secondary) fields of view, horizontally (30° and 95°) and vertically.
- The spatial frequency of peripheral vision (10% of foveal acuity).
- A random number indicating the observation ability of the observer relative to the average of the population. This is a “normally” distributed number with $m = 0.5$ and cutoff of [0,1].

Finally, the calculation depends on the particular situation:

- Range to target.
- Visibility of target. This depends on the target aspect, target posture, and occluding obstacles. Multiple observer-to-target visibility checks allow arbitrarily fine determination of occlusion. We currently use three parts for IC targets—head, torso, and legs.

The algorithm proceeds as follows:

- I. Determine the range to target (*RANGE*).
- II. Calculate the target’s critical dimension (*CD*). This is a linear, rather than areal, measure of size used in the NVESD model.
 - A. If no part of the target is visible, acquisition status is *INVISIBLE* and the algorithm ends.
 - B. *Target_Visible_Area* = area of projection of visible portion of target against a plane perpendicular to the line of sight between the observer and target. This step makes prone ICs more visible from the side than from in front, etc.

- C. *CD* is taken to be the square root of the *Target_Visible_Area*. (Lind 1995, p.15) defines characteristic dimension as the target’s “smallest” size in meters. However, since we characterize the human figure in different postures in terms of areas rather than dimensions, our model generates a linear measurement from area.

- III. Determine if the target is in the primary and secondary (peripheral) fields of view:
 - A. Primary field of view—target is within a horizontal angle (*PFOV_HORIZ_ANGLE*) and vertical angle (*PFOV_VERT_ANGLE*) of observer.
 - B. Secondary field of view—target is within a horizontal angle (*SFOV_HORIZ_ANGLE*) and vertical angle (*SFOV_VERT_ANGLE*) of observer.
 - C. If the target is not in either field of view, acquisition status is *INVISIBLE* and the algorithm ends.
- IV. Calculate the target’s angular size (*TAS*). This is an approximation based on the fact that

$$\tan(\theta) \cong \theta$$

when θ is small. For recognition and identification, and for detection in the primary field of view,

$$TAS = CD / RANGE$$

For detection in the secondary field of view, also consider target motion:

- A. Calculate Axial Motion Factor (*AMF*), which is movement along observer’s line of sight—“looming.”

$$AMF = C_1 \times |(Current\ size - last\ size)| / dt$$

where C_1 is a sensitivity constant. Note that if a target suddenly becomes visible (“blinks on”) in the field of view, the *AMF* will be non-zero for the next sighting check.

- B. Calculate Perpendicular Motion Factor (*PMF*) which is movement perpendicular to observer’s line of sight:

$$PMF = C_2 \times [CD \times V_p]^2 / dt$$

where C_2 is a sensitivity constant and V_p is the speed perpendicular to the line of sight.

$$C. TAS = (CD + AMF + PMF) / RANGE$$

V. Calculate the number of cycles (i.e., sensor elements in the eye) that overlap the critical dimension of the target.

A. Determine the spatial frequency (SF) of the sensor. The NVESD model accounts for target-background contrast, sky-to-ground brightness, atmospheric attenuation, and scene brightness by modifying the effective spatial frequency of the sensor. Our model ignores these factors for now and uses the spatial frequency that corresponds to optimal conditions. From (Lind 1995, p.24),

$$SF_{Primary} = 2.299 \text{ cycles/milliradian}$$

$$SF_{Secondary} = 0.10 \times (SF_{Primary})$$

B. Calculate cycles on the target for primary and secondary field of view:

$$N = TAS \times SF$$

VI. **Detection.** If the target has already been detected, skip to Recognition. Otherwise:

A. Calculate the probability of detection given infinite time (P_{inf}) for each applicable field of view. The function, from (Lind, 1995, p. 15), is based on the ratio between the number of cycles on the target (N) and the number of cycles required for an average observer to have a 50% probability of acquisition (N_{50}).

$$P_{inf} = [N/N_{50}]^E / [1 + (N/N_{50})^E]$$

where $N_{50} = 1.0$ (Lind, 1995, p. 11) and E is given by

$$E = 2.7 + 0.7 \times N/N_{50} .$$

B. Compare P_{inf} to the observer's acquisition ability ($ABILITY$). Unlike the NVESD model, which generates a random number for each observer-target pair, we use one number per observer. If the test fails, this target will never be detected under current conditions, and the check ends. If the target is in the secondary field of view, detection is instantaneous and the algorithm proceeds to Recognition. Otherwise calculate the time required to detect in the primary field of view (T_{det}) in seconds:

$$1. T_{det} = (-3.4 / P_{inf}) \times \ln (1.0 - ABILITY)$$

While the NVESD uses a new random number in this expression for each sighting attempt; we use the $ABILITY$ which is constant for the observer.

2. If the target has been in the primary field of view for T_{det} time, then the acquisition status is DETECTED; otherwise it is just VISIBLE and the algorithm ends. The NVESD model also makes sure that T_{det} is less than the time it takes to search the entire field of view. We have chosen not to do this, but in the future we plan to reformulate the detection time equation to account for field of view size.

VII. **Recognition.** If the target has already been recognized, skip to Identification. Otherwise, calculate P_{inf} for recognition for each applicable field of view and compare it to $ABILITY$. N_{50} for recognition is taken to be 3.5 (again, Lind, 1995). If the test succeeds, then this target's acquisition status is immediately set to RECOGNIZED.

VIII. **Identification.** Calculate P_{inf} for identification for each applicable field of view and compare it to $ABILITY$. N_{50} for identification is taken to be 6.4. If the test is successful then this target's acquisition status is immediately set to IDENTIFIED.

Figure 2 illustrates typical values generated by this model for an IC application. The bottom and top

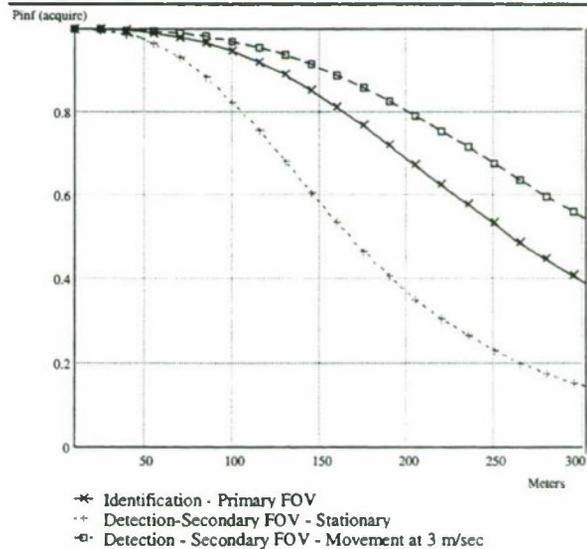


Figure 2. Probabilities of detection and identification of a standing human, vs. range.

curves show the probabilities of detecting a human when it is in the secondary field of view. The top curve is for a moving person. The center curve shows the probability of subsequently identifying the human when the primary field of view is turned toward it. Intuitively, these probabilities seem too high to us, even for excellent lighting conditions; it is possible that the probabilities should be reduced to account for visual clutter in the scene.

4. Sound

4.1 Sound Generation and Propagation

As with visual stimulus generation, sound generation and propagation is complex and dependent on many physical phenomena. A first order model of sound is actually more difficult to create than a simple visibility model. The simple visibility model assumes all objects reflect a lot of light to the observer, and thus the object can be seen if there is a clear line of sight to it. Object size, which is information easily obtained, can be the dominant factor affecting detection probability. In typical battlefield environments, sound dissipates over much smaller ranges than does light, so the intensity of the source and the effects of propagation are much more important. The sound generation characteristics of an entity are complex and not readily derivable from visual models. Entity sound characteristics depend on what the entity is currently doing. Sound propagation depends not just on the line of sight, but on reflections over many paths.

We have not attempted to build a high fidelity sound generation and propagation model. Not only would a high fidelity model be computationally expensive, but it would depend on experimental data that we do not have. Instead, we have attempted to build a simple model that captures—at least qualitatively—some tactically significant phenomena.

Our sound generation model assumes that the primary source of sound for humans is the foot hitting the ground, and that the sound generated by footfalls is proportional to running speed. In addition, the sound level is reduced for ground types that are not hard (dirt, grass, etc.). The maximum sound level while running, heard at 1 meter, is a configurable parameter. For vehicles, we assume that the primary source of sound is the engine, and that engine noise is proportional to engine load. We estimate engine load by comparing the vehicle's current acceleration value with the maximum acceleration allowed:

$$\text{Sound_Level} = \text{Vehicle_Max_Sound} \times \text{Acceleration} / \text{Max_Available_Accel}$$

where *Vehicle_Max_Sound* (sound level at 1 m) is a configurable parameter for the vehicle. *Max_Available_Accel* is a function of speed in our vehicle dynamics model, e.g.

$$\text{Max_Available_Accel} = \text{Max_Accel} \times (1 - \text{Speed} / \text{Max_Speed})$$

Weapon fire is also important. Each entity generates sound at a certain level when it fires its weapons. This level is also configurable. For all sound sources, we assume that the sound level is equal in all directions from the source.

Our sound propagation model is currently very simple: sound is attenuated with increasing distance and absorbing and reflecting surfaces are ignored. Attenuation is $20 \times \log(\text{Distance})$ in dB, which was estimated from (Woodson, 1981). In addition, we reduce the sound level 10 dB if there is no clear line of sight from source to listener. We do not currently model propagation time, as sound can cross our terrain database in about a half a second.

4.2 Sound Detection

Once sound reaches the listener, audio detection phenomena must be modeled. Characteristics of human hearing include a limited active dynamic range and the masking effect of louder sounds, adaptation to different sound levels, ability to discriminate frequencies, ability to discern the direction of the sound, and aftereffects of loud sounds. We are chiefly concerned with masking effects and direction determination in our model. At any moment, we compute the loudest sound reaching the listener and allow him or her to detect that sound and any others within 30 dB. This range was estimated from the results of masking experiments, described for example in (Geldard, 1953). We allow the listener to determine the location of the sound source exactly. This is perhaps the biggest weakness of our model. However, this simplification avoids a great deal of computation and it matches the current localization capability of the manned TTES trainee platforms. In some cases it may not even be unrealistic, as humans in the real world may be able to use other cues to localize sound.

As with visual detection, we model continuous audio phenomenon with discrete sampling. Since weapons fire is truly asynchronous in DIS, we accumulate fire events between samples.

5. Internal Situation Model

5.1 Partial Identification

As indicated in the sections above, our CGF uses the fairly standard (e.g. Lind 1995) entity identification levels "detected" (entity presence known, but nothing known about it), "recognized" (class of entity--vehicle, lifeform, etc. is known), and "identified" (everything about the entity is known). In our sound detection model, footsteps and engine noise provide recognition; small arms weapons fire provides identification.

These identification levels are coarse aggregations of facts known about a target and do not really represent knowledge well. For example, it is not clear how useful the distinction between detected and recognized is for ICs. "Identification" is not really identification of a specific individual, which could be useful at the IC level. Ideally, all bits of information about an entity could be determined from inference as well as observation. Information includes all publicly observable facts present in an Entity State PDU plus other information such as whether the entity is a unit leader, whether it is damaged, what it is carrying, what its age is, how well trained it is, etc. Various observations could fill in different facts about the entity. For example, the behavior of a soldier could indicate that it was a unit leader; the motion of a soldier could indicate what load it was carrying; or the path of a vehicle across rough ground could indicate that the vehicle is tracked. This is an area for future work.

5.2 Situation Memory

ICs in particular must remember entities that they no longer see. Their field of view is limited and entities disappear if the observer turns around; in addition, most threats may be out of sight behind cover most of the time.

We have developed a representation for an entity's mental model of the entities it has seen. When entities are currently visible and detected (to some level), they are "real." Entities that have been detected with sight or sound but are not still visible

are given a status of "figment." A complete record of information known about them when last seen is recorded. The positions of figments can be tracked by sound if they continue to make noise. If an observer looks at a location thought to contain a figment but the figment is not observed there or elsewhere, the figment becomes a "ghost." This ghost is known to exist but the observer does not know exactly where it is. (Possible locations may be inferred.)

When an entity is detected, goes out of sight, and then reappears, the observer must determine if it saw the same or different entities. On one hand, there are many details of appearance, equipment and weapons carried, and motion that might allow an observer in the real world to distinguish between individuals. On the other hand, if such discrimination is not possible in the real or virtual world, it could be arbitrarily difficult to determine how many individuals were seen. Sophisticated constraint reasoning would be required to estimate the true situation (e.g., how many individuals were seen at once? Could one individual have moved between the observed locations in the time observed?).

6. Conclusion

Individual combatant simulations involve a great deal of detail and require relatively high fidelity models, especially in urban environments. It is common in such environments for trainees to interact with CGF entities at ranges under 10 m, unlike in armored combat environments. Behavior in the IC environment from moment to moment often depends on visual and audio sensing. This sensing is primarily natural human vision and hearing. We have adapted existing CGF detection models and created new ones to provide our IC CGF with realistic visual and audio sensing abilities and mental situation models. This has allowed us to create much more realistic IC behavior.

7. Acknowledgment

This work is being supported by contract N61339-94-C-0006 from the Naval Air Warfare Center Training Systems Division.

8. References

Geldard, F. (1953) *The Human Senses*, 2nd Edition, John Wiley and Sons, Inc.

- Haber, R. N., and Hershenson, M. (1980) *The Psychology of Visual Perception*, 2nd edition, Holt, Rinehart and Winston.
- Lind, J. (1995) *Target Acquisition Models for Janus (A)*, NAWCWPNS TM 7811, Naval Air Warfare Center Weapons Division.
- Rajput, S., Karr, C., Petty, M. and Craft, M. (1995) "Intervisibility Heuristics for Computer Generated Forces," in *Proceedings of the Fifth Conference on Computer Generated Forces and Behavioral Representation*, University of Central Florida.
- Woodson, W. (1981) *Human Factors Design Handbook*, McGraw-Hill, Inc.

9. Authors' Biographies

Douglas A. Reece is a Computer Scientist at the Institute for Simulation and Training. He is the Principal Investigator of the TTES Computer Controlled Hostiles project. His research interests are in artificial intelligence, specifically intelligent agent design and computer vision. He has a Ph.D. in Computer Science from Carnegie Mellon University and B.S. and M.S. degrees in Electrical Engineering from Case Western Reserve University.

Ralph Wirthlin is an Associate Computer Scientist at IST. He has worked on individual combatant simulation for about a year and a half. Before coming to IST Mr. Wirthlin was a Research Associate at the Institute for Defense Analyses.

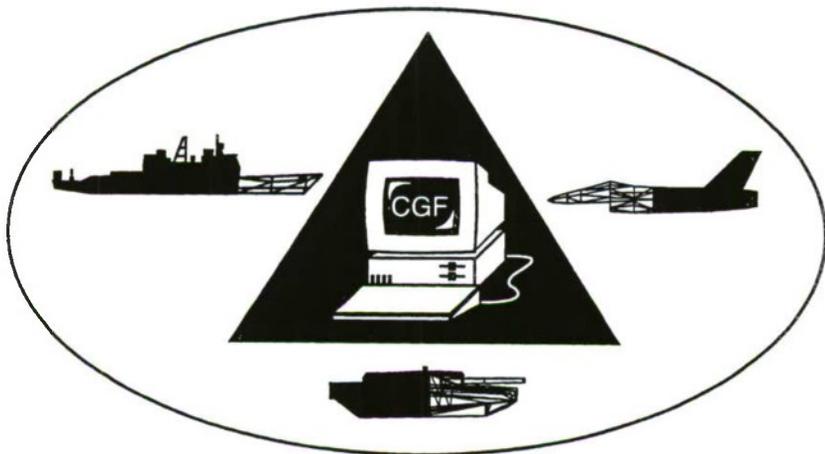
Session 6b: Systems Architecture

Howard, UT, Austin

Schricker, UCF/IST

Ullom, NAWC AD, Patuxent River, MD

White, GMU, CS Dept



Indirect Fire Support Simulation on the ModSAF Virtual Battlefield

Martin D. Howard
Applied Research Laboratories, The University of Texas at Austin
Post Office Box 8029
Austin, Texas 78713-8029

1. Abstract

The Modular Semi-Automated Forces (ModSAF) simulation has historically placed heavy emphasis on platoon and company level maneuver entities operating in a close battle environment. These behavioral representations are supported by observation, enemy detection, terrain recognition, tactical movement, and direct fire engagements, all of which are suitable behaviors to operate in the task frame environment on which ModSAF is based. However, the introduction of entity-based indirect fires into the virtual battle has required significant conceptual changes in the manner by which combat simulations are employed and driven.

Current ModSAF functionality incorporates fire support through a series of graphical user tools to plan and implement indirect fires. Application of these tools require highly skilled operators to effectively utilize these tools to create indirect fires effects on the virtual battlefield, which supports the reality of the maneuver scenario being executed. While this may lend a degree of realism to the maneuver personnel using ModSAF, it lends little training benefit to the fire support personnel employed in support of the operation. In fact, it presupposes a high level of expertise, since this fire support operator must effectively simulate the resultant behavior of all echelons of fire support personnel who would normally be supporting the maneuver force in the exercise, as well as simulating the responsiveness of the fire units.

Effective simulation of indirect fire support must take into account the dissimilarities by which these assets conduct themselves in support of a tactical operation. The tactics employed by fire support assets differ greatly from their maneuver counterparts, since they do not engage in close combat operations. Hence, the maneuver task frame and action drill behaviors so prevalent in ModSAF lends little application to fire support entities. The actual engagement of maneuver units in close combat relies on direct observation and reaction to enemy deployments and activities, and is largely conducted by individual entities. However, indirect fire support relies upon information exchange (tactical messaging) between stationary elements of a structured command and control organization to

provide target acquisition, fire support coordination, and supporting fires. Also, fire support missions are short in duration and event driven, which does not lend itself to the time-driven simulation world found in ModSAF.

The Applied Research Laboratories, the University of Texas at Austin (ARL:UT), together with the Defense Advanced Research Projects Agency (DARPA) and the U.S. Army Simulation, Training, and Instrumentation Command (STRICOM), have been developing an entity-based fire support simulation in ModSAF in conjunction with the Command Forces (CFOR) program. This effort necessitated significant design changes to allow the event-driven fire support simulation to work within the ModSAF environment and allow the leveraging of existing tactical rulesets required to simulate fire support entity behavior. Recent integration of the initial phases of this work into the ModSAF baseline shows that this design approach is feasible, and may be explored as an avenue to support other event-driven behaviors. This paper will present an overview of this simulation concept, and introduce the design approach selected to serve as the infrastructure for future development of tactical mission behaviors.

2. Current Virtual Fire Support Capabilities

Recent years have brought an explosive growth in the application of computer simulation technology to provide high-quality, cost-effective training to military combat personnel. These applications range from combat crew trainers to extensive "man-in-the-loop" wargaming systems, which create a virtual battlespace in which entity models may exist and interoperate within a defined framework. Although other virtual battle simulations exist, ModSAF essentially may be viewed as the standard virtual simulation envisioned for use in training command and staff groups in battlefield operations and tactical control measures.

The virtual battlefield currently defined within ModSAF consists of representations of terrain and combat entities, and a suite of tools used to establish the guidelines within which the entity models operate. These tools are used to refine the basic (default) behaviors of virtual entities and to create/link

task frames for the entities which define the tactical operations each shall conduct. Using these tools, the user may create tactical scenarios in which created entities maneuver according to a derived plan, detect opposing entities, and engage them with direct fires within the limits imposed by the system user. These limits consist (basically) of the ability to control whether a particular entity will engage a target on acquisition (freeing weapons to fire) and specifying a level of proficiency.

Hence, the current system supports the visual (line of sight) requirements of the close maneuver battle and possesses a fairly mature ability to control the tactical disposition and displacement of combat entities. However, this represents only one facet of a tactical operation. Success depends every bit as much on the incorporation of other assets into the overall scheme of battle. Maneuver forces cannot hope to succeed without indirect fire, air defense, engineering, signal, and logistical support. Some tools do exist within ModSAF to provide a limited functionality in some of these areas, but they are derived from a maneuver perspective and do little to train ModSAF users either in the unique missions of these supporting arms, or more critically, in the coordination of efforts between the combat and support elements essential to overall success. Effective operational coordination (both in planning and execution) should be the cornerstone of any command and staff training objectives.

Although shortcomings may be noted in several areas, as indicated above, this paper focuses on the necessity of incorporating indirect fire support capabilities into the virtual battlefield environment. In general, the task frames and behaviors provided in existing ModSAF baseline releases are not suitable for fire support entities. Existing behaviors were developed to simulate the direct fire battle and do not address the combat operations and tactical behaviors of the Field Artillery units and sections supporting the indirect fire battle. Some behaviors have limited similarity, but only through very careful and constant manipulation can a highly skilled fire support individual use ModSAF to approximate a simulation of integrated fire support. The level of skill required to affect this indirect fire behavior tends to suggest that no fire support training benefit would be derived by using the system in this fashion.

For example, given a tactical scenario running on the current ModSAF where a friendly Field Artillery unit is sustaining damage, a skilled operator could represent a hasty displacement of a Field Artillery

unit by quickly creating a unit movement task frame for the selected unit, defining a route to another location, and immediately initiating the move. However, if properly implemented, a hasty displacement to an alternate position should be a reactive measure that a Field Artillery unit should initiate in response to receiving a significant volume of incoming indirect fires on their position. This reactive behavior should occur without operator intervention or direction.

ModSAF does provide some fire support tools which may be used to simulate indirect fire support. These tools do have benefit in that they may effectively be used to provide pseudo-indirect fire effects in the virtual environment to enhance the realism of training maneuver command and staff personnel. The Fire Support Editor can be used to simulate fire missions, but the tool is operator intensive, and again requires significant, pre-existing skills to use the tool properly, thus precluding a need to use ModSAF to train the operator. There is also a munitions implementation button, affectionately known as the "finger of God", which allows instantaneous detonation of any pre-selected munition load. This tool is unrealistic in that it allows instantaneous response with unrealistic effects.

Section 4 will introduce the specific capabilities that need to be considered to make a suitable virtual environment to support the indirect fire battle. Before considering these needs, a brief description of the concept of fire support (the tactical activity which provides indirect fires) is useful.

3. Overview of Fire Support

3.1 General

During combat operations, maneuver forces cannot be expected to deal with all targets which present themselves. The threats present to maneuver forces range far beyond those opposing maneuver elements which are within range of visual capabilities. Other threats include opposing indirect fire systems, observation locations, command and control centers, assembly point for reserve forces, and logistical centers. All these classes of threat are ideal targets for indirect fires. Also, maneuver commanders in defensive operation may elect to engage targets with indirect fires to preclude giving away the location of a friendly unit prematurely.

All maneuver brigades are employed in battle with indirect fire assets (cannon artillery) which are in direct support of the brigade. These systems augment the organic mortar assets included with the maneuver forces. Additional, longer-range artillery assets [e.g., Multiple Launch Rocket System (MLRS)] are available in general support to division and corps elements. Additionally, maneuver elements (brigade and higher) may be supported by air assets (rotary and/or fixed wing) and Naval Gunfire.

Potential targets are identified by intelligence sources or by sensor systems. This information is forwarded to Fire Support Elements (FSE), which are attached to each maneuver echelon from company to corps. The FSEs determine which potential targets are attackable, and prioritize that list for consideration. These targets are then scheduled for attack and a determination is made with respect to what type indirect fire system will be used in the attack. Targets may be scheduled for attack immediately, or, in the case of preplanned targets, scheduled in accordance with a time sequence or as "on call" missions. Those targets scheduled for attack by cannon and rocket units are forwarded to a Field Artillery Command Post (CPs), where specific units to engage the targets are determined.

Fire support is conducted in two distinct phases; planning and execution. Planning encompasses those activities that are performed prior to the actual conduct of a tactical operation. At that point, the execution phase begins. Planning functions are grouped logically and tend to be viewed as process-oriented, while execution functions are more task-oriented, in keeping with the precepts of the U.S. Army's Training Evaluation Program (ARTEP). Five functional areas have been defined which comprise indirect fire support.

3.2 Fire Support Planning (FSP)

This functional area is the heart and soul of fire support, and consists of those processes which are used to determine the suitability of a maneuver plan in terms of fire support's ability to support the maneuver forces by fires, to determine the guidance by which subordinate units will conduct the execution phase of the plan, and to determine a schedule of indirect fires against planned targets which will pose an immediate threat to the maneuver forces during execution. The products of this process is the Fire Support Plan, the Field Artillery Support Plan, and the Fire Support Execution Matrix.

3.3 Fire Support Coordination and Control (FSCC)

Although two distinct functional areas, Fire Support Coordination and Fire Support Control are combined for ease of discussion and understanding. FSCC deals with those tasks associated with the delivery of fires during the execution phase of the operation, and is commonly referred to as fire mission processing. There is a tendency to view the tasks of Fire Support Coordination as being performed by Fire Support Elements (FSEs), while Fire Support Control tasks are performed by Field Artillery units. However, it is better to view Fire Support Coordination as a process that produces a target and determines a suitable means to attack it, while Fire Support Control is the process by which the mission is conducted after those selections are made. The sequential performance of the tasks associated with FSCC is what lends the process flavor to these functional areas. FSCC tasks are also often logically grouped in terms of these sequential events for a single class or category of fire mission, which are commonly referred to as fire mission threads.

3.4 Movement Control (MC)

This functional area comprises those tasks that are required to position and emplace fire support assets in positions to support the initial execution phase of the operation and to control the movement of those assets after inception of the operations. Movement during execution consist of both planned and reactive movement. Unlike maneuver elements, fire support assets normally travel in column formations from one stationary position to another, and do not normally require provisions for individual maneuvering in reaction to contact. This may change as the Paladin and Crusader systems assume the principal place in the definition of movement doctrine, but the standard maneuvering schemes used by mechanized and armor assets will still not be applicable to fire support formations.

3.5 Tactical Operations (TACOPS)

This functional area is a repository for a collection of miscellaneous tasks performed in support of the execution phase of an operation. These tasks are normally performed by support elements rather than the indirect fire units themselves. Missions performed within this area include ammunition resupply,

meteorological operations, survey operations and status reporting.

4. Requirements of the Indirect Fire Battle

Conceptually, there are fundamental differences in the direct fire and indirect fire battles, which must be addressed within the virtual training environment. The close maneuver battle is more execution intensive, allowing individual combatants the ability to react to the immediate threats identified (line of sight acquisition) within their environment. As discussed previously, ModSAF was largely developed to provide a suitable virtual environment to support this. Within the fire support community, there is a limited need to observe the enemy directly, with the exception of those observation elements which are attached to forward maneuver forces and who are responsible for identifying targets of opportunity and responding to requests for fire support from the supported maneuver commanders.

In contrast, indirect fire support strives to provide adequate and responsive fires in a target-rich environment in the face of a highly lethal counterfire threat. Hence, effective analysis of target intelligence and appropriate planning to support maneuver forces with indirect fires during execution of their operation is the goal of those persons responsible for indirect fires. To accomplish this, the fire support community relies heavily on automated command and control measures, using a structured, data-intensive communications network, to move targeting information efficiently about the battlefield and to provide the fires necessary to defeat identified targets.

The current ModSAF environment provides very limited support for structured command and control decision making during either the planning or the execution phases of an operation. Existing entity behaviors are very reactive in nature, and do not exhibit more than a rudimentary ability to assess information and to make intelligent choices during execution. Maneuver elements, when confronted with an opposing entity, engages the target with a selected organic weapon, and does not even consider indirect fires or air support as a means to defeat the target.

In contrast, indirect fire support entities, at all echelons, rely on the ability to collect and analyze information to determine the most effective and efficient means to service targets and maneuver requests for fire. The indirect fire environment is target rich and asset poor, so selection criteria must

be clearly defined and logically checked to maintain tactically realistic results. Each mission considered will, at each echelon, result in numerous available options.

The current ModSAF is essentially a time-driven simulation, with state transitions and activities occurring within specific time slices which recur steadily. This environment suits the reactive nature of the maneuver entities, which display rapidly change states, such as changing locations during movement, tube orientation, and ballistic flyouts. Although an implementation of indirect fire simulation could be designed to operate in a like manner, an event-driven simulation is more appropriate to model the processes by which fire support is accomplished. Fire support assets normally operate in stationary positions, until forced to move by the tactical situation. This move is then conducted in a deliberate fashion, and existing ModSAF functionality will probably be suitable to support this.

Fire support assets react to trigger events which instantiate sequential steps in the overall fire support mission processes which define the fire support tactical mission. Normally, but not always, these trigger events are in the form of a tactical message received via the communication network. Although the current ModSAF does include provision to send messages between entities, often the interaction between entities is determined by shared access to a common database and special utilities coded to represent verbal or visual communications.

In summary, it should be clear that an effective virtual simulation of the indirect fire support battle should allow for intelligent entities reacting to events which trigger a logically structured decision process. To support this, the entities should be capable of routing data between them via a structured communication network. Although not present in the ModSAF baseline, work has been accomplished to establish a working prototype event-driven simulation and structure communications network within the ModSAF environment in support of the Synthetic Theater of War (STOW) Command Forces (CFOR) effort.

5. FS CFOR Implementation Overview

5.1 General

The Fire Support Command Forces (FS CFOR) project was initiated to begin providing enhanced indirect fire support capabilities to maneuver forces operating in the ModSAF virtual environment. FS CFOR seeks to create intelligent command entities (CE) that can apply interpretive logic to situations and operational orders to make tactical decisions and plan, direct, and influence the behavior of subordinate semi-automated forces within the virtual environment. In order to accomplish this, CEs receive information through both traditional ModSAF techniques and the transmission and receipt of Command and Control Simulation Interface Language (CCSIL) messages. CCSIL messages are intended to simulate, in the virtual battlefield, the voice and data message traffic present on the live battlefield.

5.2 Decision Support Logic

The FS CFOR effort grew out of existing fire support simulation technology and the behavioral representations which were completed for the Fire Support Automated Test System (FSATS) program. FSATS is an ongoing test instrumentation effort by ARL:UT which incorporates an event-driven, message-based simulation which has proven highly effective in support of operational testing of the Advanced Field Artillery Tactical Data System (AFATDS). This program has invested a great deal of time and resources to define the logical behavior, in the form of tactical rulesets, associated with the various fire support command and control players to support the processing of routine fire missions.

In the FSATS simulation, software objects are created which represent simulated players, or operational facilities (OPFACs) in the test environment. Each object maintains an independent suite of tactical state information, which is used in determining appropriate courses of action. The objects communicate via actual tactical networks using appropriate protocols and data formats. On receipt of a tactical message, the receiving model determines a course of action based on its internal state and the information contained in this triggering event. Once the appropriate pathway is determined, the resultant activities dictated by that pathway (normally updating state, building and transmitting an output response message, and entering a "wait for" state) is accomplished. The generic process by which this occurs is graphically depicted in the following figure.

This basic simulation concept and generic model was leveraged directly into the FS CFOR effort. The simulation engine which was designed for the FS CFOR effort is known as the Decision Manager, and versions of this functionality run currently both in ModSAF and FS CFOR software.

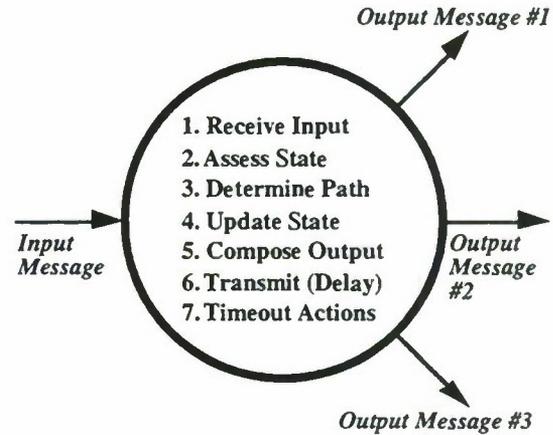


Figure 1: The OPFAC Logic Model

5.3 Abstract Message Formats

Inherent in this simulation approach is the knowledge that a sound inter-object communications network must exist. In the FSATS system, live tactical networks, which supported the wide array of fire support messages was used. The resulting FSATS simulation remains effective, but the numerous differing representation of the four major messaging schemes supported by FSATS [AFATDS Variable Message Format (VMF), TACFIRE Bit-Oriented Messages (BOM), TACFIRE Character-Oriented Messages (COM), and TACFIRE Fixed Format] led to an implementation where decisions were based in large part on the system with which the simulation was interfacing. This has proven cumbersome to maintain, as changes to decision logic and utility software must be made to accommodate routine changes to tactical messages.

In the FS CFOR effort, it was determined that this approach should not be used, and that the impacts of external message formats and data representation should be isolated away from the Decision Manager. Happily, the overall CFOR effort had already implemented the CCSIL concept to emulate tactical messaging. This gave ARL:UT the ability to define the Fire Support Abstract Message Set (FSAMS). This internal data representation forms the basis for the fire support CCSIL messages (implemented by the MITRE Corporation) and maps to messages used

FIRE REQUEST	FM INFO/CONTROL	INTELLIGENCE REPORT
FR;GRID/POLAR/LASER/SHIFT FR;MOV1/MOV2 PTGT;RPT (Firefinder FM;CFF) FM;CFF MM_FSE_FR RO MM_CP_FR RO MM_OTF RO MM_CP_FO RO MM_FO RO	FR;QUICK FOCMD MTO;TGT EOM;SURV FM;FOCMD FM;MTO FM;EOM FM;MFR MM_Commands RO MM_In_Progress RO MM_EOM RO MM_MFR RO MM_Coord_Request RO MM_Coord_Response RO MM_Status RO	ATI;GRID/POLAR/LASER/SHIFT SHELREP ATI;CDR ATI;AZR ATI;SHELREP MM_ATI RO
UNIT DATA	AMMO DATA	MOVE RECORD
OBSR;LOC RDR;LOC AFU;UPDATE UM_Basic_Unit UM-Unit_Detailed	AFU;AMMO AFU;CSR AFU;AMOL UM_Ammo_Summary UM_Ammo_Detailed UM_Cn_Mtr_Munitions UM_Rocket_Missile_Munitions UM_Propellant UM_Fuze MM_EOM RO	AFU;UPDATE AFU;SR UM_Move

Figure 2: Fire Support Abstract Message Set (FSAMS)

by the live devices to ensure a complete translation (when required). A summary chart of the current FSAMS messages, and their equivalent tactical messages is shown in Figure 2. At the current time, only the Fire Request and the Fire Mission Information/Control messages have been implemented.

5.4 Integration of Live Elements

When defining the FSAMS message contents, care was taken to ensure that each message contained a sufficient amount of data elements to allow a mapping of the data contained in the CCSIL messages to the various formats used by the live devices in the fire support community. Then using a modified version of FSATS, which has been adapted to be compatible with the Distributed Interactive Simulation (DIS) environment, ARL:UT was able to create a prototype interface box which will allow integration of live fire support players with simulated entities existing in the CFOR virtual world. This effort, known as the AFATDS-DIS bridge, is

operational in prototype form, and contains sufficient functionality to allow the incorporation of a live Brigade FSE, using AFATDS, to interact with CFOR players in conducting an end-to-end fire mission.

6. Current State of FS CFOR

The FS CFOR development has completed its initial prototype phase. At the current time, five fire support entities have been developed which support basic fire mission processing (fire for effect mission thread). These entities are the Forward Observer (FO), the Fire Support Team (FIST), the Battalion (BN) FSE, the Mortar Platoon, and the Field Artillery Direct Support Cannon Platoon.

The FO surveys its area of reference in the virtual environment and determines whether opposing force entities entering into that area are to be classified as targets, according to its guidance tables. If so, a fire request is prepared and forwarded through the FIST to the BN FSE. This entity determines whether the

target should be engaged with mortars or cannon assets and forwards the fire request to the selected entity. The fire mission then runs to completion, with the entities creating appropriate message traffic and transmitting these via the CCSIL network.

Continuation of this work should receive a high priority, if FS CFOR is to become an effective tool to train key fire support command and staff elements. Additionally, a complete, entity-based fire support virtual simulation should be considered as a strong candidate to drive collective training exercises in the near future.

7. Author's Biography

Martin D. Howard is a 1979 graduate of the United States Military Academy and a past U.S. Army Field Artillery officer with several years experience in automated command and control systems. Currently, Mr. Howard is the Program Manager of the Fire Support Automated Test System (FSATS) for Applied Research Laboratories, The University of Texas at Austin. Additionally, he acts as a domain engineer, defining simulation architectures and the tactical behaviors used to model Fire Support command and control processes.



An Architecture for Linking Aggregate and Virtual Simulations

Stephen A. Schricker, Robert W. Franceschini, David R. Stober, Jonathan C. Nida
Institute for Simulation and Training
3280 Progress Drive, Orlando, FL 32826
sschrick@ist.ucf.edu

1. Abstract

Computing power has often been the primary factor limiting the number of entities that a stand-alone, virtual-level Computer Generated Forces (CGF) system can support. By connecting several CGF systems to a network, using a communications protocol such as Distributed Interactive Simulation (DIS), CGF developers have been able to increase exercise capacity. However, the capacity of virtual exercises is still relatively small; networked CGF systems that use DIS are currently restricted to exercises on the scale of hundreds of entities. Yet large-scale exercises, consisting of perhaps tens of thousands of entities, are desirable both for their ability to teach cooperative techniques to large groups of soldiers and for their potential to visualize analysis scenarios at the entity level.

One way to extend such exercise limitations is to integrate one or more aggregate-level wargame simulations into the virtual environment; the wargames provide the context for a large-scale exercise, while smaller conflicts are played out in the virtual environment. In addition, CGF entities can supplement manned simulators—and even live equipment—to populate the virtual battlefield.

IST has developed an interface architecture that links multiple aggregate-level wargame simulations to multiple virtual components in DIS. In this architecture, each aggregate and virtual component uses a Simulation Interface (SI)—a set of functionality that allows interaction across the Aggregate+Virtual (A+V) boundary—to manage the communications between the linkage components.

This paper presents IST's architecture for linking multiple aggregate-level wargame simulations to multiple virtual components. It discusses the issues that make such a linkage difficult, and the communications required to support the architecture. This paper also presents details of the architecture's implementation as part of two projects: Integrated Eagle/BDS-D, which links the Eagle aggregate simulation to the IST CGF Testbed; and

Eagle/ITEMS, which links Eagle to ModSAF and ITEMS—both CGF systems.

2. Background

Computer-based battlefield simulations may be classified into two broad categories: *aggregate* and *virtual*. In aggregate simulations, the typical atomic simulation object is the military unit; the simulation abstracts the soldier- and vehicle-level details of the battlefield so that it can model higher-level issues. For example, an aggregate simulation would represent a mechanized infantry company as a single object rather than as a group of battlefield entities consisting of dismounted infantry, armored personnel carriers, and tanks. The aggregate simulation maintains all of the characteristics for the unit as a whole and performs statistical analyses on the unit's actions to determine its overall state. In contrast, the typical atomic simulation object in a virtual simulation is the battlefield entity. Given the example above, a virtual simulation would represent the mechanized infantry company as several instances of dismounted infantry, armored personnel carriers, and tanks. To gather any unit-level details regarding the company, the virtual simulation must make inferences based on the actions of the individual entities that belong to the unit.

In addition, aggregate and virtual simulations often treat the passage of time differently. Virtual simulations are typically used for battlefield training, which requires strict adherence to the passage of time; the realism of the virtual simulation depends upon the exercise's events occurring at a rate that corresponds to that of real-world events. Thus virtual simulations are considered *real-time* simulations. Aggregate simulations, on the other hand, are typically used for analytic purposes, such as tactical situational evaluation. The time that it takes for the aggregate simulation to process the events for a particular time interval is not necessarily related to the amount of actual time that the interval represents (were those events to occur in reality). Thus, most aggregate simulations may be considered *non-real-time*; they may run faster or slower than real-time,

depending on the resolution of the events that they model.

The differences in entity and time representation between aggregate and virtual simulations creates difficulties in simulation interoperability. For example, in the virtual environment, it is difficult for the individual battlefield entities of a virtual simulation to detect and react to aggregate units. Similarly, units in aggregate-level wargame simulations do not typically detect and react to individual battlefield entities. The problems associated with differing time representations are obvious; the simulations need to operate on the same time scale in order for their interactions to make sense (Karr 1994).

The architecture presented in this paper addresses many of these interoperability issues.

3. The Aggregate+Virtual (A+V) Architecture

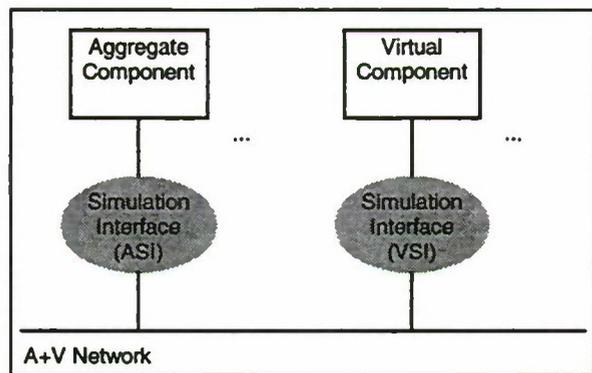


Figure 1: Conceptual A+V Architecture

Figure 1 shows a conceptual diagram of the A+V linkage architecture developed by IST. A typical A+V linkage would consist of one or more aggregate components in combination with one or more virtual components. An aggregate component is an aggregate-level wargame simulation, such as the Eagle simulation developed by the TRADOC Analysis Center (TRAC). A virtual component is a CGF system possibly networked with manned simulators or live simulations (Stober 1995b). For example, a single virtual component might consist of a ModSAF suite along with several manned simulator or live simulation resources; the manned/live simulations provide a training environment for soldiers, while the ModSAF suite supplies additional friendly- and opposing-force entities.

Each aggregate and virtual component uses a Simulation Interface (SI) to manage the

communications among the linkage components. The SI provides the functionality that allows interaction across the A+V boundary.

3.1 Simulation Interface (SI)

The Simulation Interface (SI) coordinates communications across the A+V boundary (Karr 1994). Each aggregate and virtual component in the linkage has its own specialized SI that is responsible only for the assets of its linkage component. Although it would be possible to construct a single, integrated SI that supports all of the components in the linkage, a distributed SI properly separates and isolates the details of each component's specific requirements. Thus, each SI services only the communication to a single linkage component. This gives SI implementers considerable flexibility—the SI can be a distinct node on the network, a complementary process on its linkage component's node, or a program module integrated into its linkage component's software (Stober 1995b).

Though this architecture will support multiple aggregate and virtual components, the remainder of this discussion will assume an A+V linkage consisting of only one aggregate component and one virtual component.

3.1.1 Functional Architecture

Perhaps the most important concept regarding this architecture is that it describes a functional rather than a physical design. In other words, it is the role of the SIs to provide a certain set of functionality to the system in order to maintain the A+V linkage. Where this functionality resides and how it is implemented are issues that are beyond the scope of this paper.

3.1.2 Aggregate SI (ASI)

The ASI provides the interface between the aggregate simulation and the virtual environment. It has two functions: it incorporates virtual-level data into the aggregate simulation, and it provides aggregate-level data to the virtual environment. The ASI gathers virtual-level data, such as entity status information and requests for action, and forwards this data to the aggregate simulation for processing. The ASI also receives unit-level updates from the aggregate simulation and places this data onto the A+V network for the Virtual SI(s) to process.

3.1.3 Virtual SI (VSI)

The VSI provides the interface between the virtual simulation and the aggregate environment. It, too,

has two functions: it incorporates aggregate-level data into the virtual component and it provides virtual-level data to the aggregate environment. The VSI gathers aggregate-level data, such as unit status information and requests for action, and forwards this data to the virtual component for processing. The VSI also receives entity-level updates from the virtual component and places this data onto the A+V network for the ASI(s) to process.

3.2 Unit States

In an A+V exercise, only one component may control a particular unit or entity at any one time. If an aggregate component controls a unit, the unit is said to be *aggregated*. If a virtual component controls a unit's entities, the unit is said to be *disaggregated* (Stober 1996). A+V researchers have defined several other unit states, such as *icon* (Franceschini 1995), *fully-disaggregated*, *pseudo-disaggregated*, *partially-disaggregated*, (Foss 1996), *gradually-disaggregated* (Cox 1995), and *locally-disaggregated* (Calder 1995), but they are essentially special cases of aggregated and disaggregated. (The exception is *partially-disaggregated*, in which more than one component may simultaneously control a unit and/or its entities. The circumstances surrounding units in this state, however, are beyond the scope of this paper. See Foss (1996) for a more complete discussion.)

3.2.1 Disaggregation

When the disaggregation of a unit is triggered, the ASI instructs the aggregate simulation to temporarily stop simulating the unit; unit updates will come from the ASI (via the virtual component and the VSI) while the unit is disaggregated. The VSI then instantiates the unit's entities on the virtual component. When the disaggregation process is complete, the ASI receives entity updates from the VSI, compiles this data, and forwards it to the aggregate simulation.

3.2.2 Aggregation

When a unit no longer needs to be disaggregated, the VSI instructs the virtual component to remove the unit's entities from the virtual environment. The ASI informs the aggregate simulation that it can resume simulating the unit, and begins providing the virtual environment with aggregate data regarding the unit.

3.3 A+V Interactions

Most interactions in an A+V exercise occur at the virtual level. There are a few instances, however,

when the aggregate and virtual environments must interact directly. When a unit is disaggregated, the virtual component requires information describing the unit's current operations. Otherwise the unit's entities would not know what actions to perform while the unit is disaggregated. Likewise, a disaggregated unit may request indirect fire from nearby artillery. The aggregate simulations are responsible for allocating the indirect fire to the appropriate artillery units.

3.3.1 Operations Orders

When a unit is disaggregated, the VSI receives from the ASI operations-order data for the unit, which it distributes to the unit's entities in an appropriate manner. In addition, the ASI receives updates from the VSI regarding the intentions of disaggregated units in response to changes in environmental conditions. This dialog persists while the unit is disaggregated.

3.3.2 Indirect Fire

The ASI receives requests for indirect fire from virtual components, which it forwards to the aggregate simulation. When the aggregate simulation allocates the indirect fire to an appropriate artillery unit, the ASI sends the resulting indirect-fire volley data to the VSI, which converts it into fire and detonation data for introduction into the virtual environment.

4. A+V Interoperability Protocol (IOP)

The A+V Interoperability Protocol (IOP) is the communications mechanism that transfers command and control information, as well as information detailing aggregate- and virtual-level unit activities, across the A+V boundary (Karr 1994). Although the functionality of the IOP is well established, the implementation of the IOP is an issue that is beyond the scope of this paper.

The IOP provides five primary message types: initialization messages establish communications pathways when components begin participating in an exercise; unit status messages allow the exchange of aggregate- and virtual-level information; state-transition messages allow units to request aggregation or disaggregation; operational details messages provide a means of communicating a unit's operations and intentions; and indirect-fire messages allow aggregated and disaggregated units to exchange indirect fire.

4.1 Initialization Messages

Every SI transmits an initialization message to inform the linkage of its presence in the exercise. The other SIs respond accordingly to acknowledge the message.

4.2 Unit Status

Unit Status messages provide the means by which the aggregate and virtual components update their local records for remotely-controlled units. The ASI receives virtual-level data from the VSI and incorporates it into the aggregate simulation, while transmitting aggregate-level data to the VSI. Likewise, the VSI receives aggregate-level data from the ASI and incorporates it into the virtual component, while transmitting virtual-level data to the ASI.

4.3 State Transition

State Transition messages allow a simulation operator or unit commander to request that a unit change its state. This permits a transfer-of-control over units across the A+V boundary.

4.3.1 Disaggregation

When the ASI receives a request to disaggregate a unit, it instructs the aggregate simulation to suspend the simulation of the specified unit and informs the VSI that it can proceed with the disaggregation. The VSI then instantiates the proper entities on the virtual component. Once the disaggregation is complete, the VSI transmits virtual-level data regarding the unit to the ASI, which incorporates the data into the aggregate simulation.

4.3.2 Aggregation

When the VSI receives a request to aggregate a unit, it instructs the virtual component to remove the unit's entities from the virtual environment. The ASI then informs the aggregate simulation that it may resume the simulation of the unit. Once the aggregation is complete, the ASI resumes transmitting aggregate-level data regarding the unit to the VSI, which incorporates the data into the virtual component.

4.4 Operational Details

Operational Details messages provide a means of communicating a unit's instructions and intentions across the A+V boundary. When a unit is first disaggregated, the aggregate simulation transmits the unit's operations orders to the ASI. The ASI sends this information to the VSI, which forwards it to the

unit commander for interpretation. As operations change or orders are completed, the aggregate simulation will transmit fragmentary orders in the same manner. In addition, the unit commander can transmit commander's intent messages to the aggregate simulation, via the VSI and ASI, in response to environmental conditions.

4.5 Indirect Fire

Indirect Fire messages allow for indirect-fire interactions across the A+V boundary. A unit commander may request indirect fire from nearby artillery. The VSI receives the request and forwards it to the aggregate simulation via the ASI. The aggregate simulation receives the request, allocates the indirect fire to an appropriate unit, and computes the details of the ensuing indirect-fire volleys. The ASI receives the volley data and forwards it to the VSI, which converts the indirect-fire volley data into fire and detonation data, and introduces it into the virtual environment at the appropriate times.

5. Difficulty Issues

There are a number of issues that make the implementation of an A+V linkage difficult. Among these are the size of the IOP messages, the implementation of direct fire across the A+V boundary, and a side effect of automatic disaggregation triggers—spreading disaggregation.

5.1 IOP Message Size

Since aggregate unit-state messages can carry data regarding perhaps hundreds of individual battlefield entities, the messages themselves can become enormous. It is important to take this into consideration during implementation. These messages may need to be separated into several smaller messages in order to satisfy message size requirements.

5.2 Direct Fire

Direct fire between aggregate units and virtual entities is much more difficult to address than is indirect fire. Current research has failed to provide not only a solution, but also an adequate definition of the problem itself. Attempts have been made to remedy the problem of A+V direct fire by ensuring that direct-fire confrontations occur in the same environment (usually virtual), but a true solution has yet to be found (Stober 1996).

5.3 Spreading Disaggregation

One benefit of A+V is that the linkage software itself can monitor the exercise and aggregate or disaggregate units as it deems necessary. For example, as a unit enters a "high-resolution" area (Karr 1994), where any action inside occurs at the virtual level, the system can automatically disaggregate the unit without human interaction. In addition, as two opposing-force units approach one another, the system can trigger their disaggregation in order to resolve any ensuing conflict at the virtual level.

An anomalous result of automatic disaggregations, though, is "spreading disaggregation," in which one disaggregation triggers a series of disaggregations (Petty 1995). This could produce a windfall effect that would eventually overload the resources of the virtual environment. This issue will need to be addressed as research into automatic disaggregation triggers progresses.

6. Implementation Details

IST has been involved in the implementation of two A+V linkages: Integrated Eagle/BDS-D, which links the Eagle aggregate simulation with the IST CGF Testbed and manned simulators; and Eagle/ITEMS, which links Eagle with ModSAF and ITEMS—both CGF systems. The following sections discuss implementation details for these two projects.

6.1 Integrated Eagle/BDS-D

The Integrated Eagle/BDS-D Project, which serves as a proof-of-concept for the interoperability of aggregate and virtual simulations, links the Eagle aggregate simulation with the IST CGF Testbed (along with manned simulators), and can operate in either SIMNET or DIS. Eagle is a UNIX process that can run on Sun or Hewlett-Packard workstations. The Testbed is a CGF system that runs on several networked IBM-compatible personal computers and consists of at least two nodes: a Simulator, which performs the entity simulation; and an Operator Interface, which provides a graphical user interface for the CGF operator.

6.1.1 Process Architecture and Message Pathways

Figure 2 shows the process architecture and message pathways for the Integrated Eagle/BDS-D system. The Eagle Simulation Interface Unit (SIU) provides the functionality of the ASI and communicates with Eagle using UNIX Remote Procedure Calls (RPC).

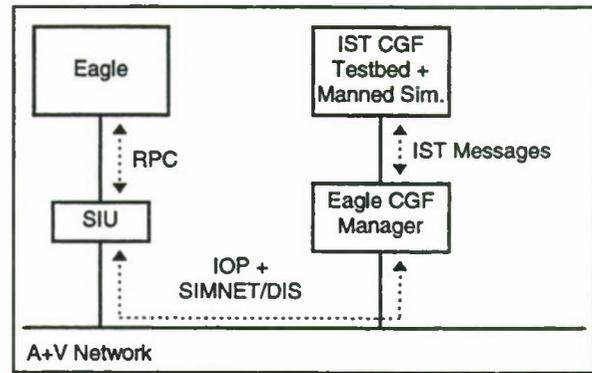


Figure 2: Integrated Eagle/BDS-D Process Architecture and Message Pathways

The Eagle CGF Manager provides the functionality of the VSI and communicates with the Testbed via special TCP/IP network packets called IST Messages.

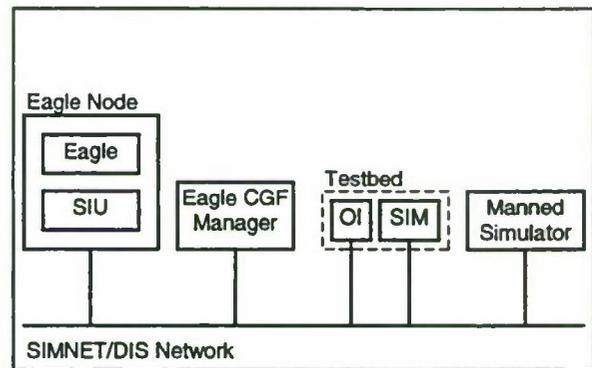


Figure 3: Integrated Eagle/BDS-D Network Configuration and Implementation

6.1.2 Network Configuration and Implementation

Figure 3 shows the network configuration and implementation for the Integrated Eagle/BDS-D system. The SIU is implemented as a distinct process that runs on the same node as the Eagle simulation. The Eagle CGF Manager is a modified IST CGF Simulator with its entity simulation capability disabled.

6.1.3 Interoperability Protocol (IOP)

The Integrated Eagle/BDS-D IOP is implemented differently for the SIMNET and DIS versions of the system. For SIMNET, the IOP consists of a combination of pre-defined SIMNET PDUs and new IOP PDUs. The SIU gathers virtual-level entity data by monitoring the SIMNET network for Vehicle Appearance PDUs; the remaining IOP messages are carried inside SIMNET Association PDUs with a special value identifying those PDUs as Integrated Eagle/BDS-D IOP messages.

For DIS, the SIU gathers virtual-level entity data by listening to DIS Entity State PDUs. The remaining IOP messages are carried inside IST Messages.

For more information regarding the Integrated Eagle/BDS-D architecture and network configuration, see Karr (1994).

6.2 Eagle/ITEMS

The Eagle/ITEMS linkage, which is part of the ongoing development of the Aviation Digitization Laboratory at Fort Rucker, Alabama, links the Eagle aggregate simulation with ModSAF and ITEMS, and operates in DIS. ModSAF is a UNIX process that can run on Silicon Graphics, Sun, or Hewlett-Packard workstations. ITEMS is also a UNIX process that can run on Silicon Graphics workstations.

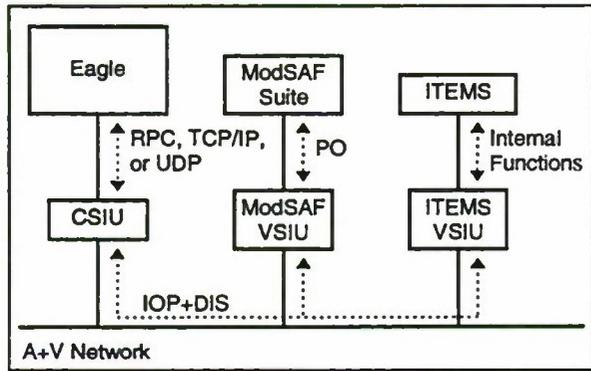


Figure 4: Eagle/ITEMS Process Architecture and Message Pathways

6.2.1 Process Architecture and Message Pathways

Figure 4 shows the process architecture and message pathways for the Eagle/ITEMS linkage. The Eagle Constructive Simulation Interface Unit (CSIU) provides the functionality of the ASI. Early versions of the CSIU communicated with Eagle via UNIX RPCs. However, more recent versions are configurable to use TCP/IP or UDP network packets—an example of the flexibility that this architecture affords its implementers.

The ModSAF Virtual Simulation Interface Unit (VSIU) provides the functionality of the VSI for ModSAF and communicates with ModSAF using the Persistent Object (PO) protocol.

The ITEMS VSIU provides the functionality of the VSI for ITEMS and, since it is implemented as a program module integrated directly into the ITEMS software, communicates with ITEMS simply via function calls.

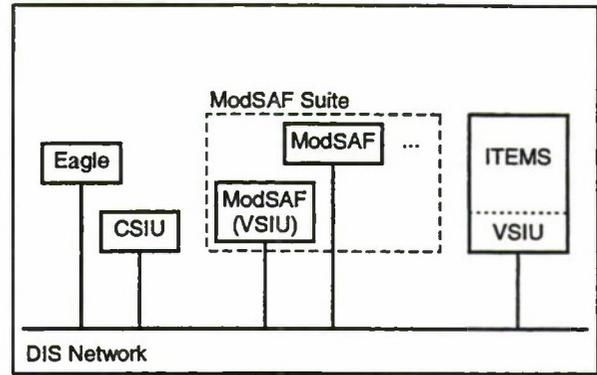


Figure 5: Eagle/ITEMS Network Configuration and Implementation

6.2.2 Network Configuration and Implementation

Figure 5 shows the network configuration and implementation for the Eagle/ITEMS linkage. The CSIU is implemented as a ModSAF Simulator that has undergone extensive modification (including the elimination of its entity simulation capability) so that it can communicate with Eagle. Thus it is a distinct process that runs on its own network node.

The ModSAF VSIU, like the CSIU, is implemented as a ModSAF Simulator, and is also a distinct process that runs on its own network node. Unlike the CSIU, however, the ModSAF VSIU retains its functionality to simulate virtual entities.

The ITEMS VSIU, as stated previously, is implemented as a program module integrated directly into the ITEMS software.

6.2.3 Interoperability Protocol

The Eagle/ITEMS IOP is implemented as a combination of pre-defined DIS PDUs and new IOP PDUs. The CSIU gathers virtual-level entity data by monitoring the DIS network for Entity State PDUs. In addition, the CSIU provides aggregate-level unit data to both VSIUs by broadcasting both DIS Aggregate State PDUs and specially developed IOP Unit Detail PDUs. This was necessary because the prototype Aggregate State PDU used during development did not meet all of this particular system's requirements.

The remaining IOP messages developed specifically for the Eagle/ITEMS linkage are carried inside DIS Message PDUs with a special value identifying those Message PDUs as Eagle/ITEMS IOP messages.

7. Conclusions

Aggregate+Virtual linkages are viewed as a solution to the current limitation of virtual exercises: the virtual network is incapable of supporting very large scale exercises. A+V is also viewed as a new and powerful tool for conducting analytic studies and as a way to bring legacy systems into the virtual environment. The architecture presented in this paper seeks to standardize the methodology for linking aggregate and virtual simulations in order to facilitate the development of similar systems in the future.

The architecture presented in this paper discusses only the functionality that an Aggregate or Virtual SI must provide to the A+V linkage. Implementation details are left to developers, giving them the freedom to tailor future A+V implementations that use this architecture to the specific requirements of the linkage components that they intend to incorporate.

8. Acknowledgment

This research was sponsored by the U.S. Army Simulation, Training, and Instrumentation Command as part of the Integrated Eagle/BDS-D project, contract number N61339-92-K-0002. That support is gratefully acknowledged.

9. References

- Calder, R.B., Peacock, J.C., Panagos, J., and Johnson, T.E. (1995). "Integration of Constructive, Virtual, Live, and Engineering Simulations in the JPSPD CLCGF", *Proceedings of the Fifth Conference on Computer Generated Forces and Behavioral Representation*, Institute for Simulation and Training, Orlando FL, May 9-11, 1995, pp. 71-82.
- Cox, A., Maybury, J., and Weeden, N. (1995). "Aggregation Disaggregation Research—A UK Approach", *Proceedings of the 13th Workshop for the Standards for the Interoperability of Defense Simulations*, Institute for Simulation and Training, Orlando FL, September 18-22, 1995, pp. 449-464.
- Foss, W.R. and Franceschini, R.W. (1996). "A Further Revision of the Aggregate Protocol", *Proceedings of the 14th Workshop for the Standards for the Interoperability of Defense Simulations*, Institute for Simulation and Training, Orlando FL, March 11-15, 1996, pp. 727-738.
- Karr, C.R. and Root, E.D. (1994). "Integrating Constructive and Virtual Simulations", *Proceedings of the 16th Interservice/Industry Training Systems and Education Conference*, Orlando FL, November 28 - December 1, 1994, section 4-5.
- Petty, M.D. and Franceschini, R.W. (1995). "Disaggregation Overload and Spreading Disaggregation in Constructive+Virtual Linkages", *Proceedings of the Fifth Conference on Computer Generated Forces and Behavioral Representation*, Institute for Simulation and Training, Orlando FL, May 9-11, 1995, pp. 103-111.
- Root, E.D. and Karr, C.R. (1994). "Displaying Aggregate Units in a Virtual Environment", *Proceedings of the Fourth Conference on Computer Generated Forces and Behavioral Representation*, Institute for Simulation and Training, Orlando FL, May 4-6, 1994, pp. 497-502.
- Stober, D.R., Kraus, M.K., Foss, W.F., Franceschini, R.W., and Petty, M.D. (1995a). "Survey of Constructive + Virtual Linkages", *Proceedings of the Fifth Conference on Computer Generated Forces and Behavioral Representation*, Institute for Simulation and Training, Orlando FL, May 9-11, 1995, pp. 93-102.
- Stober, D.R. and Franceschini, R.W. (1995b). "Simulation Interface Designs for the Aviation Digitization Laboratory", *Technical Report IST-TR-95-13*, Institute for Simulation and Training, July 14, 1995.
- Stober, D.R., Schricker, S.A., Tolley, T.R., and Franceschini, R.W. (1996). "Lessons Learned on Incorporating Aggregate Simulations Into DIS Exercises", *Proceedings of the 14th Workshop for the Standards for the Interoperability of Defense Simulations*, Institute for Simulation and Training, Orlando FL, March 11-15, 1996, pp. 391-396.

10. Authors' Biographies

Stephen A. Schricker is an Associate Computer Scientist at the Institute for Simulation and Training, and currently leads the software engineering effort for IST's participation in the development of the Aviation Digitization Laboratory at Fort Rucker, AL. He has earned a B.S. in Computer Science from the University of Central Florida, and is currently pursuing an M.S. in Computer Science at UCF. His research interests include simulation, networking, and fuzzy logic.

Robert W. Franceschini is a Computer Scientist at the Institute for Simulation and Training. He

currently leads the Integrated Eagle/BDS-D Project at IST. Mr. Franceschini has earned a B.S. in Computer Science from the University of Central Florida and is currently pursuing an M.S. in Computer Science at UCF. His research interests are in the areas of simulation, graph theory, and computational geometry.

David R. Stober is a Research Consultant on the Integrated Eagle/BDS-D Project at the Institute for Simulation and Training. He has earned a B.S. in Computer Science from the University of Central Florida, and is currently pursuing an M.S. in Computer Science at UCF. His research interests are in the areas of simulation and artificial intelligence.

Jonathan C. Nida is a Graduate Research Assistant at the Institute for Simulation and Training. He has earned a B.S. in Computer Science from the University of Central Florida and is currently pursuing an M.S. in Applied Mathematics at UCF. Mr. Nida's research interests include simulation, parallel computation for mathematics, and dry nanotechnology.

Using An Ordnance Server to Provide Validated Weapon Models to ModSAF[†]

Larry Ullom
Aircraft Simulation Branch
Naval Air Warfare Center Aircraft Division
48140 Standley Road
Patuxent River, MD 20670-5304
lullom@msis.dmsomil

Peter Fischer
J.F. Taylor Inc.
P.O. Box 760
Lexington Park, MD 20653
pfischer@jfti.com

1. Abstract

The Modular Semi-Automated Forces (ModSAF) simulation tool traditionally models a weapon's physical and behavioral characteristics by loading simplistic algorithms that are derived from generic data into the main simulation application. This causes inherent problems. The processor load of the weapon models must remain relatively low to maximize the entity count per workstation. This is often the leading cost driver in an exercise. In addition, using sensitive weapon data imposes security issues on the ModSAF application.

The operational requirements for certain training exercises dictate that weapon models must perform as a functionally valid replica of the actual system. This ensures the training performed would enhance the trainee's performance in a similar situation. Given the current approach, it is too computationally expensive to add validated models to the actual ModSAF application. The use of an Ordnance Server (OS), as demonstrated by the Air Combat Environment Test and Evaluation Facilities' Manned Flight Simulator (ACETEF/MFS), provides a better solution.

The Ordnance Server is an external host that models weapons surrogates. Validated weapon models are incorporated into the Ordnance Server and the corresponding ModSAF models are disabled. This approach improves scalability,

provides a more level playing field between interacting entities, and segregates sensitive or classified modeling and data.

This paper will discuss the origin of the Ordnance Server concept and the process of integrating an Ordnance Server with ModSAF. An analysis of the test implementations will show the benefits of this approach. The paper will also include a discussion of open issues such as in flight guidance input from the launching entity. Finally a conclusion that looks ahead to future implementations will be provided.

2. Introduction

Traditionally the modeling of a weapon's physical and behavior characteristics was done in the model that was responsible for launching the weapon. This arrangement was the natural environment in systems that were designed to execute as a monolithic simulation on a single host. This type of system is highly limited in a number of important areas including scalability and multiplicity of reuse.

The ModSAF system was designed to address some of the problems in monolithic computer generated forces (CGF) systems. It uses distributed interactive simulation (DIS) protocols to allow multiple simulation hosts to cooperatively interact in a unified synthetic battlespace. However the concept of retaining

[†] This paper is declared a work of the US Government and is not subject to copyright protection in the United States.

ownership of all simulation attributes spawned by the ModSAF application was retained.

To further enhance the capabilities of CGF systems such as ModSAF, it is necessary to allow hand-off of simulation entities and attributes to other more specialized simulators. While this is explicitly addressed in the new Department of Defense (DoD) High Level Architecture (HLA), it is also possible to gain the benefits of using validated munition models with ModSAF in a DIS environment. The use of an Ordnance Server can provide this capability.

3. Technical Overview

The Ordnance Server extends the idea of distributed simulation by separating the simulation of the launching vehicle from the munition simulation. The concept could be applied to any pairing of munition and launch vehicle simulations. However this paper shall

consider the ModSAF launcher only. In order to understand how the Ordnance Server can be used with ModSAF it is necessary to first discuss its internal workings.

The Ordnance Server operates using only standard DIS protocol data units (PDUs). When a cooperating launch vehicle simulation wishes to fire a munition, it issues a fire PDU as it normally would. The Ordnance Server, having been previously configured to look for fire PDUs from a specific (site, application, entity), will try to match the weapon type and fusing data to a weapon it is configured to simulate. If a match is found, the Ordnance Server will instantiate a simulation of that weapon using target data from the fire PDU. The Ordnance Server issues entity state PDUs for the instantiated munition during its delivery to the target. When the fuse model indicates the termination of the munition, the Ordnance Server generates a detonation PDU.

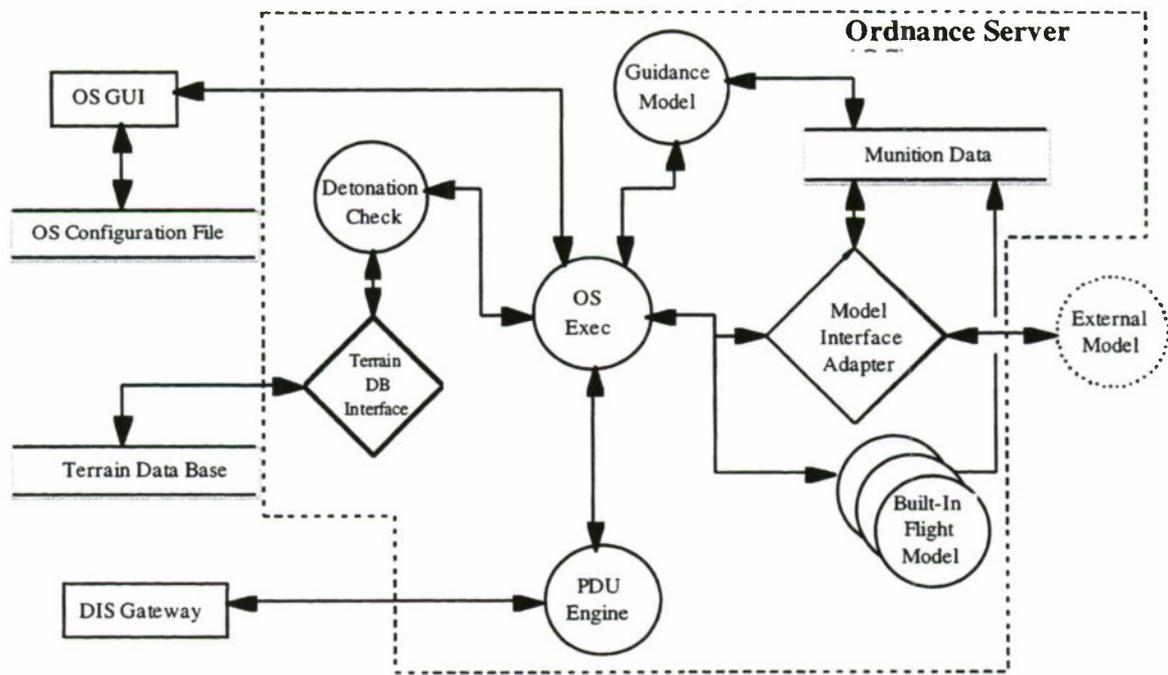


Figure 1: Ordnance Server Interfaces

3.1 Special Interfaces

The Ordnance Server supports two interfaces to external objects. These interfaces and their relationship to the rest of the Ordnance Server is illustrated in figure 1. The first is a Model

Interface Adapter (MIA). This is a code wrapper that goes around an external weapon model and provides all the translation services needed to make the model look like an internal simulation to the server executive. At the same time, the MIA simulates the environment the external model was designed to operate in. This

architecture provides a mechanism for interfacing nearly any legacy model. In particular, validated models from accepted training systems can be used to support a DIS exercise.

The second interface is the ground truth database interpreter. This provides the OS with a consistent representation of ground truth regardless of the actual format used in the underlying terrain database. The only parameter that is important to most weapon models is the height above the terrain at the point where the munition is currently flying. Other parameters could be included in this interface if a particular model required them for normal operation.

3.2 Configuration Parameters

The Ordnance Server must be configured to operate with a "parent" launch entity. The parent entity is denoted by the (site, application, entity) triplet. The Ordnance Server can serve multiple hosts by allowing the entity or entity/application identifiers to be wild cards. The entity type of any munitions to be surrogated must also be specified and mapped to a specific weapon model. Any weapon model loaded can be mapped to any munition type. Other configuration information that must be supplied includes the terrain database file name, DIS exercise and IP numbers, and types of runtime feedback desired. There are model specific parameters associated with each weapon simulation as well.

4.0 Integrating ModSAF with the OS

Integration with the Ordnance Server first required that the ModSAF internal weapon dynamics models be disabled. This resulted in relatively simple modifications because of ModSAF's highly modular design. To suppress entity state and detonation PDUs, libmissile was modified so ModSAF would only issue the Fire PDU. This allows the Ordnance Server to describe the trajectory and detonation event of the munition. A command line switch "Generate Missiles" was added to the ModSAF executable. When "-nomissiles" is specified, ModSAF's missile simulations are suppressed. Also, liblauncher was modified to removed the munitions ID number from an internal list of local entities in ModSAF. Without the id number in it's local list, ModSAF will recognize the Ordnance Server's missile as a viable remote

entity. Otherwise ModSAF would still act as if the entity were local, resulting in no icon display for the munition on the Plan View Display.

Another problem was that the designated target id was not originally specified in the Fire PDUs produced by ModSAF. This information is needed for the Ordnance Server to work correctly. The intended target was originally kept internally in ModSAF, so the change consisted of simply passing this data to the routine used to broadcast ModSAF's Fire PDUs.

Next a ground truth database interpreter based on the ModSAF libctdb services was added to the Ordnance Server. This allows both applications to share identical representations of ground truth.

4.1 Ordnance Server Advantages

The ordnance server can be used to provide models, that are accepted by the subject mater experts in a given exercise, with no penalty to the ModSAF application's processor load. These accredited models can be classified (if required) without affecting ModSAF's unclassified status. Maintaining multiple models for a particular munition at different levels of detail or different classification levels is also facilitated by this approach. Additionally a well designed network topology can reduce latency impacts on entity interactions, by collocating ordnance servers with the targets they are likely to engage.

The ordnance server has successfully been used to complement CGF systems in many large scale exercises. These exercises include the Strategic Theater of War (STOW) Engineering Demos, Navy Kernel Blitz fleet training event, and IITSEC 95 DIS Demo. In it's earliest use, a single ordnance server was used by one site to simulate only Air-to-Air missiles initiated by a single man in the loop system. Since then, many additional validated models have been added to the ordnance server and CGF systems have been modified to allow the ordnance server to simulate their munitions. As more applications use the ordnance server to simulate their munitions, the benefits to the goal of a fair fight become more apparent. Simulations that use the ordnance server will not create munition simulations with unrealistic flight or guidance attributes.

Due to the large number of entities simulated in a STOW exercise, the ordnance server's ability to

further distribute processing load is especially useful and even necessary if high fidelity, real time munition models are an exercise requirement. The flexibility of the manner in which the ordnance server can be used to do this has been demonstrated. A local ordnance server has been used to simulate munitions launched by an application located at a remote site. Multiple ordnance servers have been used by one application, each one simulating different types of munitions for the same set of entities. This characteristic of being flexible to the needs of a particular exercise scenario or hardware configuration has proven to be especially useful due to the variant nature of DIS exercises.

4.2 Open Issues

There are still open issues to be resolved with this approach. One of the major concerns regards tightly coupled systems, where the launch platform and the munition depend on either a one way or bi-directional link to function properly. An example is the Navy's SM-2 which receives steering input throughout flight from the launcher based on the launcher's radar track.

One solution to this would be to implement the link data via signal PDUs. This would increase the fidelity of the simulation. However it would also cost network bandwidth. Another solution would be to locate the sensor model on the Ordnance server. It is not clear how this could be facilitated under current DIS protocols, but the HLA fully supports this method.

Another issue is the loading of prelaunch data from the launching platform into the weapon model. Currently this is accomplished via the graphical user interface (GUI). The signal PDU is not a natural choice for this data as it would be passed via internal busses on the actual platform. One possible solution is the set data simulation management PDU. It could be used to initialize a weapon that requires this type of data prior to launch then the weapon would simply be attached to the launch platform until the fire PDU. This would require more extensive modification of ModSAF to support.

5. Conclusions

The ordnance server approach answers the problem of providing validated models for ModSAF by using an external host to take over

the flyout of weapons launched by the application to the intended target. This allows ModSAF to use appropriate weapon models for the exercise event with no additional configuration management or piecemeal code integration. The models integrated in this manner do not have to be reengineered to fit within the ModSAF architecture, and multiple models of the same munition can be substituted easily.

The concept of the use of an ordnance server to supplement CGF applications has evolved into a tested, working product. The benefits of such an approach have been demonstrated through multiple DIS exercises. As more munition models are added to the ordnance server and the existing models capabilities are further refined, the munition simulations for all applications that use the ordnance server are improved.

Further research should provide answers to the open issues discussed in this paper. It is apparent that the advantages of this solution warrant continued development.

6. Acknowledgment

The authors wish to thank the following people for their contributions to this effort: CDR. Peggy Feldmann, Brett Dufault, John DiCola, Alexandra Wachter, and David Mutschler.

7. References

- DiCola, J, Fischer, P, Mutschler, D, Ullom, L (1996) "Improving Munition Simulation Fidelity Through Use Of An Ordnance Server", *Proceedings of the AIAA Flight Simulation Technologies Conference* (to be published)
- DiCola, J, Mutschler, D, Ullom, L, Wachter, A (1996) "Providing Common Munitions Models Via an Ordnance Server" *Proceedings of the 18th Interservice Industry Training Systems and Education Conference* (to be published)
- Foster, L, Feldmann, P (1995) "The Limitations of Behavior for Valid Distributed Interactive Simulation" *95-13-027 Position Papers of the 13th DIS Workshop on Standards for the Interoperability of Distributed Simulations*

8. Biographies

Lawrence C. Ullom is a Senior Electronics Engineer with the US Navy's Naval Air Warfare Center Aircraft Division. He holds a BSEE from West Virginia Institute of Technology. He has been involved in numerous DIS working groups and demonstration projects. His interests are Networking, Simulation, and Distributed Processing.

Peter Fischer is an Electrical Engineer employed by J.F. Taylor, Inc. and working on site at the ACETEF/Manned Flight Simulator in the Naval Air Warfare Center at Patuxent River, MD. He received his bachelor's degree from George Mason University and is currently pursuing a master's degree in Computer Science from the Florida Institute of Technology. His interests include simulation, artificial intelligence, and computer graphics.



Interfacing External Decision Processes to DIS Applications

Elizabeth L. White, Kenneth E. Frosch, Vincent P. Laviano, Michael R. Hieb, J. Mark Pullen
Department of Computer Science and C3I Center
George Mason University
4400 University Drive, Fairfax, VA 22030
white@cs.gmu.edu

1. Abstract

The Distributed Interactive Simulation (DIS) protocols (IEEE 1995) allow heterogeneous applications to share exercise information. As DIS applications mature and transition to the High Level Architecture (HLA), their utility can be increased by integrating them with external decision processes. External decision processes comprise a wide range of applications such as inference methods, learning algorithms, and collaborative planning tools. There are several possible approaches to accomplishing such an integration. We analyze various criteria to identify the best architectural approach for a specific set of requirements. The criteria discussed are distribution of processes, data representation, interprocess communication, and quality of service. This paper describes three approaches and presents two case studies in constructing interfaces to the Modular Semi-Automated Forces (ModSAF) CGF simulation. By conducting a more formal analysis of the interface design process, we provide system designers with the basis to make a more informed choice when designing an interface.

2. Introduction

DIS-compliant simulations have been extensively utilized in recent automated warfighting exercises. Interfacing external decision processes to these simulations can improve their behavior and make them more widely applicable. The interface of an external decision process to a DIS-compliant simulation may take one of several forms. We have identified several criteria for designing such an interface of non-DIS applications to DIS simulations:

- 1) distribution of processes — describes the degree to which the processes are distributed, ranging from a single centralized executable to processes distributed over a network.

- 2) data representation — determines the format the processes use to communicate information.
- 3) interprocess communication — the method that the processes use to communicate, for example, shared memory or message passing.
- 4) quality of service factors — describe requirements for distributed system latency, security and robustness.

Section 3 covers these issues in detail. From these criteria we identify three general approaches to interfacing external decision processes to a DIS application. Section 4 describes two case studies as examples of two of the approaches. In these case studies, the DIS application with which we integrated our external processes was ModSAF. Further details about ModSAF are given in Section 4.1. We compare our approaches with related work in Section 5 and analyze the impact of the HLA on our integration methods. Conclusions are presented in Section 6.

3. Interface Design Issues

Several issues must be considered when interfacing an external process to a DIS application. Among those are the four criteria enumerated above. In addition to these technical factors, another factor that may influence design decisions is the existence of components, such as libraries implementing an Application Program Interface (API). Such components constrain the choices available for the architecture, however their use can also save considerable time and effort. For example, COMPASS provides libraries for performing a Remote Procedure Call (RPC) between a client application (ModSAF in our case) and the COMPASS server.

Table 1 lists the issues that need to be addressed with possible options based on the quality of service factors.

Issue	Option
Degree of Process Distribution	Single process
	Multiple processes, single host
	Multiple hosts, single subnet
	Multiple hosts, multiple subnets
Data Representation	Standard format
	Ad hoc format
IPC	Shared Memory
	Remote Procedure Call
	Message Passing

Table 1: Design Issues and Options

3.1 Degree of Process Distribution

One of the most fundamental decisions that must be made when designing the software architecture for an integrated application is the degree of distribution of the constituent processes. The level of distribution will have a great effect on the other issues of the design, particularly the IPC mechanism.

The two applications can be integrated into a single process (Figure 1), eliminating the need for interprocess communication, or the integrated application can be split into multiple processes (Figure 2), providing additional flexibility, but necessitating interprocess communication. If these processes reside on a single host, then this additional flexibility is limited, but there is a wide selection of interprocess communication mechanisms and several

data representation issues can be avoided because a homogeneous environment is guaranteed.

If the processes reside on multiple hosts, then the characteristics of the intervening network become an issue (Figure 3). The network may be a local area network (LAN) in which case there will be high reliability and little latency, but the geographic scope of any DIS exercise run using the integrated software will be severely limited. On the other hand, the application may be distributed across a wide area network (WAN), in which case there will be less reliability and increased latency, but the geographic scope of an exercise will be virtually unlimited. These characteristics may vary depending on whether the WAN is private (and can be dedicated to the DIS exercise in question at the time it is running) or public (and having to support a great deal of unrelated traffic at the same time the exercise is running).

3.2 Data Representation

Data representation is an important issue because the external process and the DIS application need to share information. Given the fact that these applications may reside on different types of hardware, any solution should accommodate heterogeneous environments. There are two ways to allow information sharing: 1) converting one of the applications to the other's data representation or 2) using a standard format as an intermediate language.

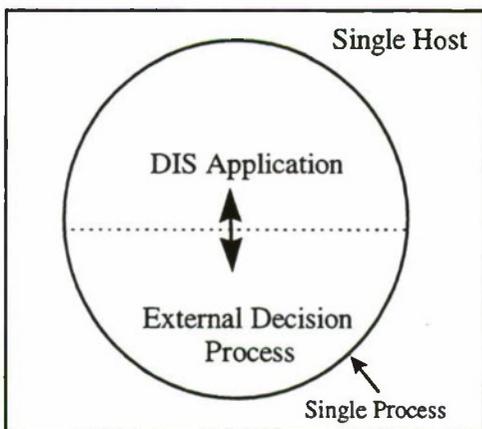


Figure 1: Single Process on a Single Host

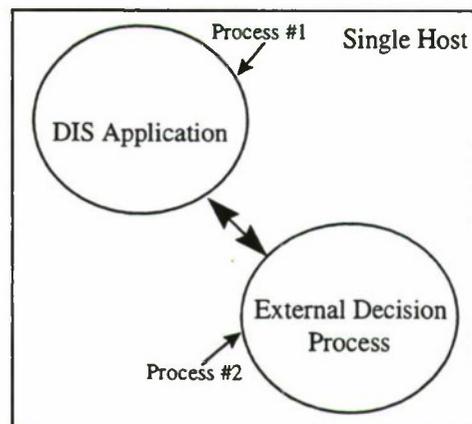


Figure 2: Multiple Processes on a Single Host

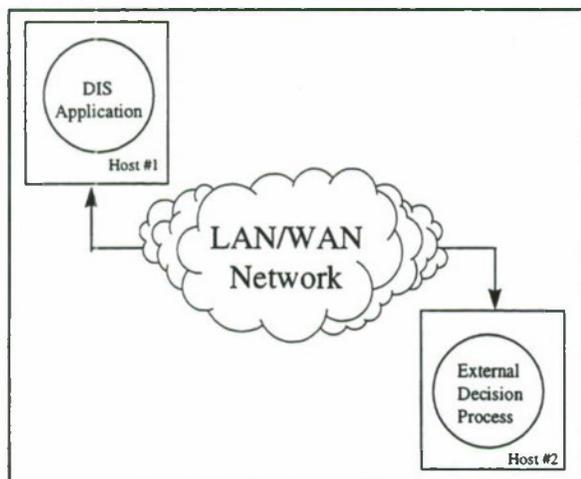


Figure 3: Processes on Multiple Hosts on a Network

The first option is often taken. The process of converting one application's data representation is often tedious and error-prone. Using a standardized format, such as the DIS protocols, does require changes to both applications and additional encoding and decoding effort at runtime. However, once an application has been modified to allow it to use the standard format, it can interoperate with any other application conforming to the standard as well.

With both options, the data must either be initially represented in both applications in some common format, or translated into a common format that both applications can interpret and manipulate. The common data format may be either a standardized format (e.g. the DIS protocols or the External Data Representation (XDR) standard used to transfer data between different machine architectures (Sun, 1987)), or an ad hoc format.

3.3 Interprocess Communication

As noted above, if the integrated application is composed of multiple processes, some form of interprocess communication (IPC) will be necessary in order to allow the processes to share data. The choice of an interprocess communication mechanism will follow largely from the choice of a process architecture (single process vs. multiple processes, distributed over a network, etc.). The need for IPC presents a design challenge because of the real-time requirement imposed by the DIS component of the integrated application. This requirement usually

necessitates the use of either non-blocking IPC or a timeout mechanism. A non-blocking IPC would allow the DIS application to continue executing while the external process services the request.

The two fundamental forms of IPC are shared memory and message passing. Shared memory is known to be faster in most circumstances (Stevens 1990), but its use is limited to applications running on a single host. Therefore, message passing will often have to be used. A popular paradigm for message passing is the Remote Procedure Call (Birrell and Nelson, 1984), in which the message passing primitives are hidden from an application programmer in linkable libraries, allowing the programmer to call both local and remote functions using the same semantics. A case study of an RPC interface is given in section 4.1.

Another possibility for interprocess communication is the distributed object technology embodied in the Object Management Architecture (OMA). Central to the OMA is the Common Object Request Broker Architecture (CORBA), which provides for transparent communication between distributed application objects (Yang and Duddy, 1996). The prototype implementation of the HLA's Run Time Infrastructure (RTI) uses CORBA. The RTI's API is specified in CORBA's Interface Definition Language (IDL) (Calvin and Weatherly, 1996).

3.4 Quality of Service

Integrated applications with a DIS component are likely to have special requirements in one or more of the following areas:

Latency. The integration of an external decision process with a DIS application implies a real-time requirement for the composite application. Meeting this requirement may be a challenge, because the external process may not have been designed with real-time performance in mind. However, if the DIS application is utilized in a mode where it is not operating in real-time, such as generation of simulation scenarios, then latency does not need to be considered. Examples are given in Section 4.

Correctness. In order to maintain the consistency of a distributed simulation it is necessary that all participants receive similar data about the simulated virtual world. This implies a requirement that some simulation data be transmitted reliably. In general,

DIS applications are designed to handle the loss or corruption of an entity state PDU, because they are sent with great frequency and each new entity state PDU supersedes previous ones. However, there are other, individual PDUs which represent important events such as collisions and must therefore be received reliably in order to ensure the consistency of the distributed simulation. Since reliable transmission of data contributes to latency, a balance needs to be struck between correctness and latency.

Robustness. Unlike a traditional centralized application, a distributed application can continue to exist after the failure of a host on which parts of the application are executing. This may entail the relocation of simulated entities (or an external decision process) to other hosts, or it may be sufficient for the simulation to continue without the entities in question and without interaction with the decision process.

Security Issues. When an application is distributed across multiple hosts, the intervening network increases vulnerability, especially if the constituent processes reside on multiple subnets separated by a wide-area network that is outside the administrative domain of those organizations conducting the exercise. Encryption and/or other forms of security may be required. The interface will need to be designed with this and other administrative concerns in mind.

Concurrent Access. It may be necessary or desirable for multiple clients to access resources managed by the integrated application concurrently. This is especially likely to be true for the external decision process portion of the application if it is implemented as a separate process. A need for concurrent access to resources will impose additional design constraints on the interface.

4. Case Studies

In this section, we first describe three basic approaches to interfacing an external decision process to a DIS application and then present detailed case studies of two of these approaches. The three basic interface approaches that we have identified are to 1) create a single executable, 2) utilize existing IPC mechanisms for distributed processes, and 3) use DIS PDUs to implement the interface.

The first approach requires integrating the DIS application and the decision process into a single executable. The Soar/Intelligent Forces (IFOR) architecture is an example of this approach. Soar provides the reasoning capabilities of intelligent automated agents for ModSAF (Laird et al, 1995, Soar/IFOR, 1996). The advantage of this approach is that no IPC is required between the two components, resulting in minimal latency. However, the disadvantages of this approach are 1) the lack of concurrency between the two components may result in an application with unacceptable performance due to the real-time demands of the DIS component, and 2) the complexity resulting from the combination of two control flows into a single flow.

The second approach is to use system-provided communication facilities for the required interactions. One of the most common interprocess communication paradigms, RPC, is supported by many underlying systems, meaning that special non-standard interfaces to the external applications need not be constructed. The advantage of this method is that the application components can be distributed across multiple heterogeneous platforms. The disadvantages of this method include additional latency introduced by local-area or wide-area network connections between the application components and the possible degradation of performance in the face of network or host failures. In addition, distributed approaches require greater attention to security.

An example of this architectural style is our interface between ModSAF and the COMPASS system (SAIC, 1995, SAIC 1996). COMPASS is the Common Operational Modeling, Planning and Simulation Strategy. The ModSAF-COMPASS interface allows planners to evaluate collaborative mission plans by using ModSAF simulation capabilities. Implementing this interface involved creating a new ModSAF library that uses the API provided by the COMPASS client library. This client library encapsulates the application-level protocol that enables the exchange of overlay and route information between the COMPASS server and ModSAF via RPC. Additional details are given in Section 4.1.

The third integration approach is to utilize DIS protocol data units (PDUs) to exchange information between application components. To do this, new PDU types must be defined and additional software to generate and process these new PDUs must be

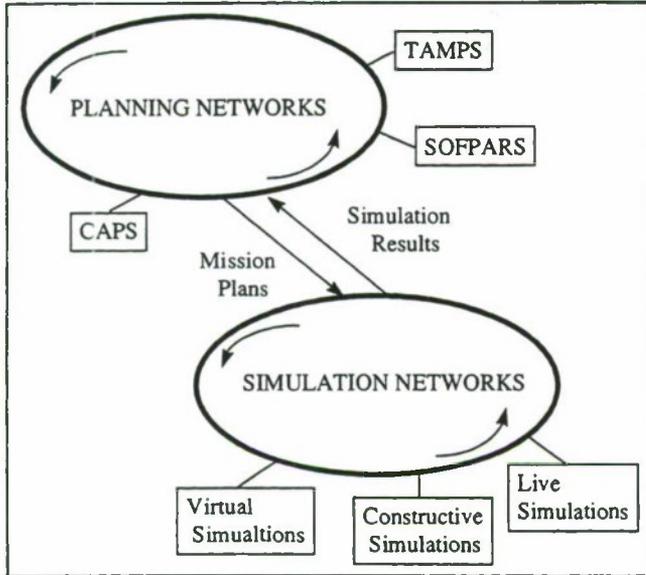


Figure 4: Mission Planning and Evaluation with COMPASS

written for both systems. In the HLA, this approach would use the RTI to communicate between the two processes. The advantages of this approach are that the application components can be distributed and that the DIS application's basic flow of control need not be disrupted. Some disadvantages of this method are that the integration is more complex than the first two approaches, and that additional software is required to make it reliable.

The DIS PDU approach is the one we have taken in interfacing the Virtual Commander (VCDR) learning functions to ModSAF. VCDR is a Lisp program that uses Disciple to perform learning for a ModSAF instructable agent (Hieb, Tecuci & Pullen 1996). In this case, we constructed an interface using experimental PDUs that transport the data necessary for the instruction process. Existing ModSAF library functions are used to send/receive the new PDU and register the appropriate callback. A DIS front end was implemented in Lisp for Disciple. The front end call functions to send and receive PDUs during the instruction process. Additional details of this integration are given in Section 4.2.

4.1 ModSAF-COMPASS Interface

In this section, we discuss the interface developed between COMPASS and ModSAF. COMPASS is a collection of distributed collaborative planning

(DCP) services that allow legacy mission planning systems to interoperate with other COMPASS-compliant mission planning systems and DIS simulation systems. COMPASS is implemented as a collection of servers providing DCP session management, shared map overlay exchange, composite route preview, and DIS-based composite mission preview. COMPASS currently supports several air operations planning systems, such as the Coordinated Adaptive Planning System (CAPS), the Special Operations Force Planning and Rehearsal System (SOFPARS), and the Tactical Air Mission Planning System (TAMPS).

ModSAF is a DIS-compliant simulation system consisting of a graphical user interface, one or more simulators, and an optional logger (Ceranowitz, 1994). ModSAF simulates entity-level actions and provides realistic terrain reasoning capabilities. ModSAF uses the DIS protocols to share information with other DIS-compliant simulation systems (Loral, 1995).

The interface constructed illustrates the potential for linking virtual simulations such as ModSAF to the COMPASS architecture. This linkage allows a collection of heterogeneous mission planning systems to access ModSAF (through COMPASS) in order to evaluate collaborative mission plans. Figure 4 shows the iterative mission planning and evaluation process using heterogeneous mission planners, the COMPASS servers, and simulators. Users of heterogeneous mission planning systems can collaborate, each using a system that is familiar to them, and create composite missions. These missions are then exported to the COMPASS servers and retrieved and executed by a variety of simulation systems. The simulation results can be fed back in the form of DIS PDUs and viewed by the participants in the DCP session. The results can be evaluated and the mission restructured to take advantage of the feedback obtained from the simulation.

4.1.1 Design Issues

We will now consider the design of the COMPASS-ModSAF interface in the context of the issues raised in section 3: data representation, degree of process distribution, interprocess communication, and quality of service requirements.

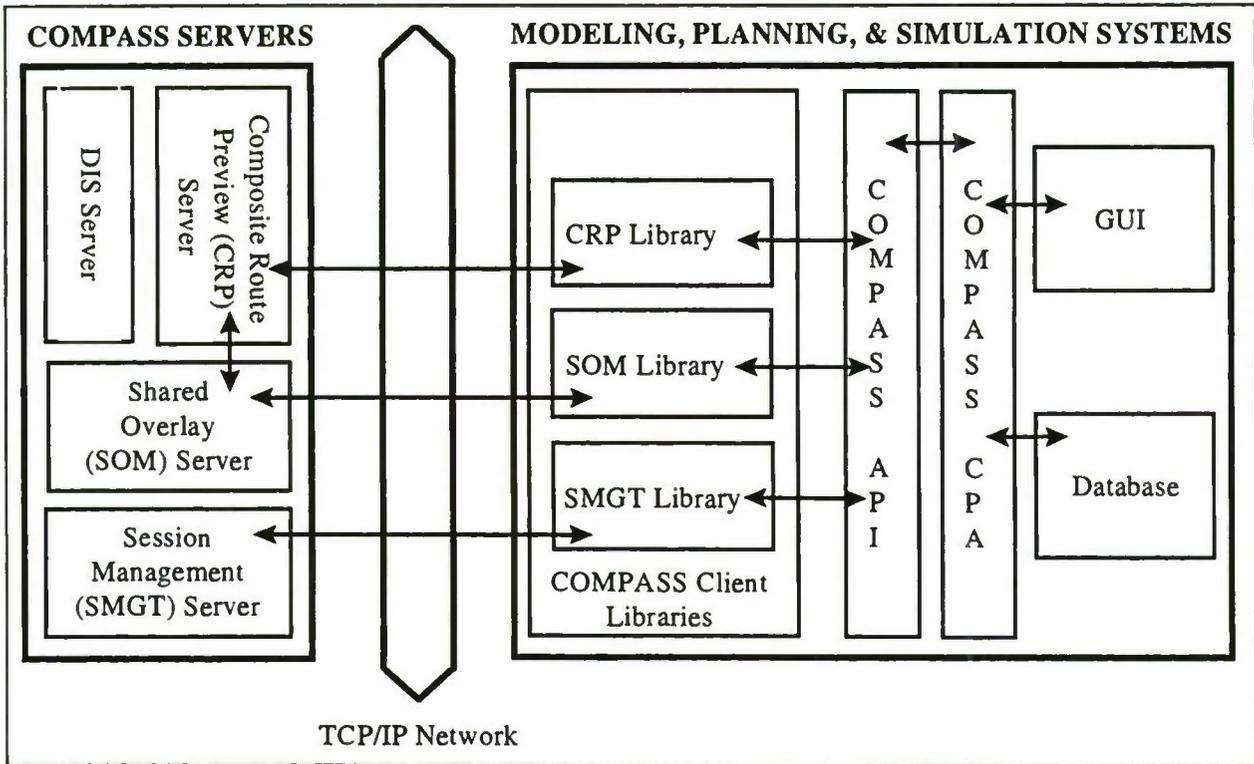


Figure 5: COMPASS Interface Architecture

As stated above in section 3.3, the choice of interprocess communication mechanism is very much based on the prior choice of the degree of process distribution. In this case, binaries for linkable libraries that implement the API function described in the COMPASS ICD were made available to us. The API functions provide an RPC facility, allowing interaction with the COMPASS servers without explicit use of message passing primitives.

The Interface Control Document (ICD) for the COMPASS Architecture (SAIC 1995) provides a detailed description of a data structure and representation to be used by COMPASS-compliant applications when communicating with the COMPASS servers. This document also describes the API through which application programs must communicate with the COMPASS servers. As these API functions expect arguments in a specific data format, the choice of data representation was predetermined. Similarly, the document describes

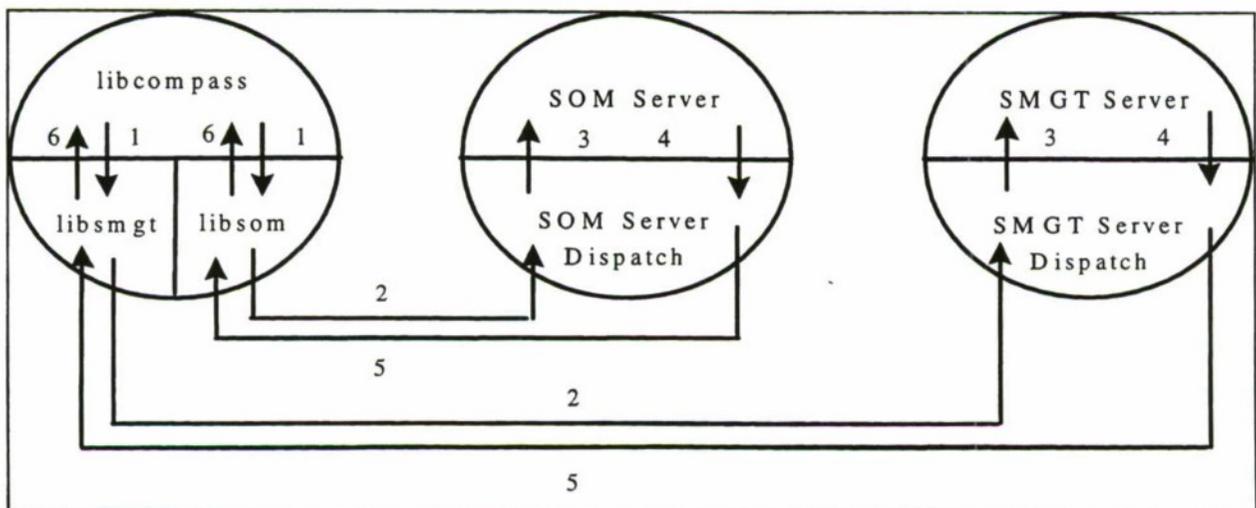


Figure 6: Remote Procedure Call in COMPASS

an interface architecture, shown in Figure 5, in which the COMPASS servers are separate processes and the collaborative planning applications communicate with them through the COMPASS libraries. Therefore, we chose the architectural approach in which ModSAF and COMPASS reside on separate hosts and communicate via a network.

An example of RPC as used in COMPASS is shown in Figure 6. Two of the COMPASS libraries, *libsom* and *libsmgt*, are linked with ModSAF at compilation time. When a ModSAF user invokes a *libcompass* function requiring a remote function be executed on one of the COMPASS servers, e.g. the SOM server, (1) *libcompass* calls a function in *libsom*. This function (2) is responsible for packaging the arguments for the server routine into one or more RPC messages along with a code indicating which remote function is to be executed and sending the message to the SOM server. When the server receives the RPC message (3), it unpacks it and determines which function to call based on the aforementioned code. The function is called with the arguments received in the message. When the server function has completed (4), the results are passed back to the dispatch function in the SOM server. (5) The results are packed into a reply message and sent back to *libsom*. (6) Upon receiving the reply message, *libsom* unpacks the results and returns them to its caller, a function in *libcompass*. Thus,

from the perspective of *libcompass*, the remote procedure has been invoked through a simple function call without the explicit use of message-passing primitives. As shown Figure 6, the same process occurs when remote functions are executed on the SMGT server.

In the absence of multiple threads of execution clients block or poll for results after sending an RPC request to the server. This makes RPC difficult to integrate into a real-time application's existing control flow, especially if the application needs to perform other tasks (such as simulating entities) and is instead waiting for data, effectively freezing the entire simulation. To address this problem, the client stub may timeout after not receiving a reply in a specified period of time, causing the RPC to fail. Still, although RPC serves to abstract away message passing primitives, it cannot be said that the message passing is truly transparent to the client application, as new forms of failure are now possible beyond those that could occur if the server procedure were being called locally.

Our interface design addresses the quality of service issues described in section 3.4. Since ModSAF is a DIS application, low latency is a very important requirement. Communication between ModSAF and the COMPASS servers takes time, especially if the COMPASS servers are located on a remote subnet.

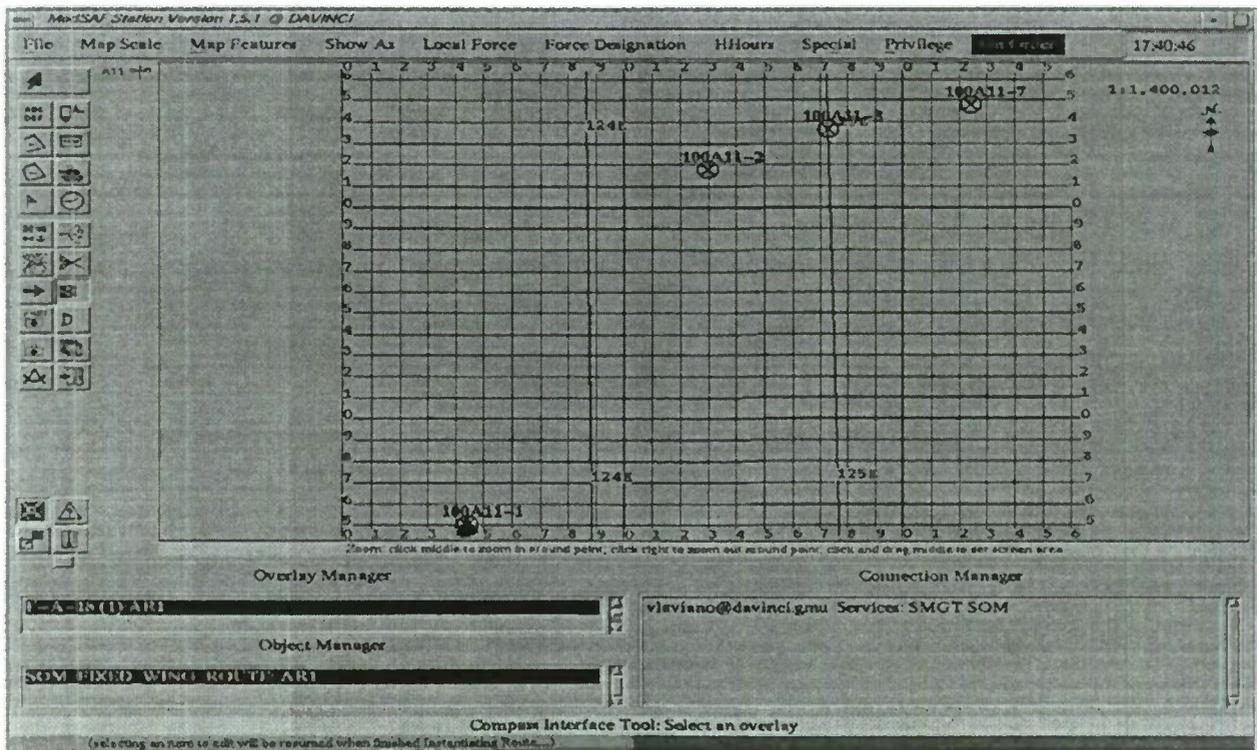


Figure 7: COMPASS Interface Tool

Because the interaction between ModSAF and COMPASS is used to establish a ModSAF simulation scenario, it takes place before any simulation activity occurs. In this case, additional latency is not a consideration because the system need not meet real-time latency requirements.

In this interface, we did not consider communication reliability or security issues because all message passing was encapsulated within the COMPASS libraries that were provided to us. Both the COMPASS servers and ModSAF are robust. ModSAF has facilities to shift entities from host to host if a host goes down via the PO database, and COMPASS maintains similar databases from which servers can recover if they crash.

4.1.2 Architecture

We now describe the architecture of the COMPASS-ModSAF interface as pictured in Figure 8. ModSAF has been designed with modularity and ease of expansion in mind, and for this reason, it consists of over 450 libraries. We decided that the best way to add to the ModSAF architecture without disrupting existing components was to create a new COMPASS interface library, *libcompass*, to isolate changes within this library

The ModSAF GUI provides several editors that allow the user to perform various tasks such as creating new entities, and adding graphical images to the map overlay. Among the additions to ModSAF found in *libcompass* is a new ModSAF editor, developed in Motif, that is accessed by pressing a new COMPASS button on the ModSAF GUI, as shown in Figure 7. The editor provides a list of users who are currently participating in the DCP session and a list of other COMPASS services these users are currently subscribed to. The editor also provides a list of shared overlays that currently reside in the COMPASS Shared Overlay Management (SOM) server's database. This information is obtained by *libcompass* through the COMPASS API functions provided in the COMPASS libraries, *libsmgt* and *libsom*, compiled with ModSAF. The editor allows a user to select an overlay. This causes *libcompass* to obtain a list of route objects contained within the selected overlay from the SOM server. The user can then click on the route object corresponding to the route to be simulated.

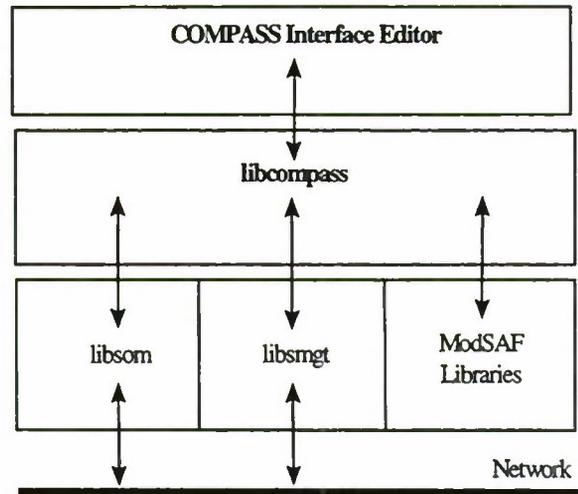


Figure 8: ModSAF-COMPASS Interface Architecture

A COMPASS route consists of an entity description, a set of waypoints comprising the route, flight characteristics which indicate when the entity arrives at each waypoint, and a set of events (each of which is associated with a waypoint).

Once a route is selected, *libcompass* maps the COMPASS entity described in the route to the most appropriate available ModSAF entity. This is done via a mapping function that makes use of an extensible mapping file that can be modified without recompilation of ModSAF. After the mapping is complete, the coordinates of the entity's starting location are converted from latitude/longitude format to topocentric coordinates (TCC) for use by ModSAF. The ModSAF entity is then created at the initial waypoint though the utility functions in *libunitutil*, another ModSAF library.

Then, all other waypoints are converted to TCC format and placed in ModSAF's persistent object (PO) database by using the functions in *libpo*, a ModSAF library relating to the PO database. A chain of tasks is then created, by using the utility functions in *libtaskutil* along with some additional code, tasking the entity to fly from waypoint to waypoint, one after another. This task is placed "On Order", meaning that simulation of the route does not begin immediately, but is pending and will begin at the time the user invokes the task.

4.2 Virtual Commander Interface

The second case study we present is an implementation of an interface between ModSAF and Disciple (Tecuci, 1988) for the Virtual Commander System. Disciple is a machine learning system written in Lisp that runs on a typical Unix workstation. It employs multistrategy learning, combining explanation-based, analogical, and example-based methods with knowledge acquisition to learn from a subject matter expert (SME).

As CGF technology advances sufficiently to allow CFORs to be created and deployed, knowledge acquisition will become a critical issue. The Virtual Commander system is an approach to solving the problem of knowledge acquisition for CFORs. VCDR is an instructable agent which utilizes Programming by Demonstration (Cypher, 1993) and Machine Learning techniques to allow a SME to teach an agent. Programming by Demonstration systems give an end user the ability to create programs by demonstrating their action.

We have prototyped this approach with the Captain system (Hieb et. al. 1995) which consisted of a file-level integration of the apprenticeship learning system Disciple and ModSAF. In order to integrate Disciple fully with ModSAF we expanded upon the Captain interface in two areas. Rather than utilizing a textual interface for the training/learning process, we are developing VCDR ModSAF editors that allow the use of the terrain map interface (plan view display) in ModSAF. To convey the data from the ModSAF editors, we are interfacing the learning functions of Disciple to ModSAF using an experimental DIS protocol data unit (PDU).

4.2.1 Design Issues

Our design and implementation is driven by three significant factors: real-time performance, data representation, and control of the learning cycle. The initial concern addressed in this implementation was the difficulty of integrating an application that relies on real-time delivery of data (ModSAF) and one that does not (Disciple).

The performance or latency issue was important since ModSAF, a DIS application, typically requires real-time transmission of data. The ModSAF/Disciple interface does not require this type of service. In fact, due to the algorithms involved in

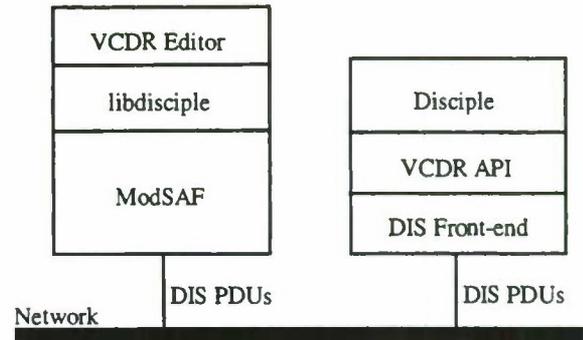


Figure 9: Virtual Commander Architecture

the various learning phases of Disciple, it would be difficult to achieve real-time response from Disciple. Therefore, the concern about real-time performance is how to construct a non-real-time communication that does not cause perceptible delay in ModSAF. We implemented this non-real-time communication by distributing the processes over a network allowing each process to utilize the full resources of their respective workstations, since both applications are very computationally intensive.

To construct this interface we delivered messages via DIS PDUs. Integrating this message passing scheme into ModSAF was done using the extensive libraries that ModSAF makes available, namely the PDU API and PDU Processing libraries. By using these libraries our messaging was integrated into the ModSAF control structure such that ModSAF executed normally. In other words, ModSAF does not block while waiting for our message, which would disrupt the flow of ModSAF's execution. This was achieved by using functionality provided by the ModSAF library, *libpduproc*, to register a callback to handle the message upon ModSAF receiving it. When ModSAF receives one of these messages it will call the registered handler function. Integrating a message passing facility into Disciple required implementing a front-end to Disciple in C that was callable from Lisp via its foreign function interface (Harlequin, 1994).

A second issue was the format in which the data would be communicated between the applications. While ModSAF stores its data in a Persistent Object (PO) database, Disciple represents its data in the form of a semantic network. In order for the two systems to communicate, a common format was constructed, described in the PDU in Table 2. The

Description	Size (bits)
DIS 2.03 header	96
Message ID	32
Fragment number	32
Learning phase	32
Message type	32
Message length	32
Message	8*length

Table 2: Virtual Commander
Experimental DIS 2.03 PDU Format

message within the PDU is data extracted from either the PO database or from Disciple's semantic network. The experimental PDU format was developed and introduced into ModSAF. By utilizing DIS PDUs to perform the messaging for the VCDR system, we can use a messaging infrastructure already established for DIS.

The final issue was the need to move the control interface from Disciple to ModSAF. Originally, the steps taken by the SME to teach Disciple were selected from a menu presented by the Disciple software in Lisp, based on which phase of the learning process was being executed. When introducing the GUI editor into ModSAF, the control mechanism was transferred to ModSAF. A new library, *libdisciple*, was built into ModSAF to implement the control features needed. The resulting interface is much more natural to its intended user.

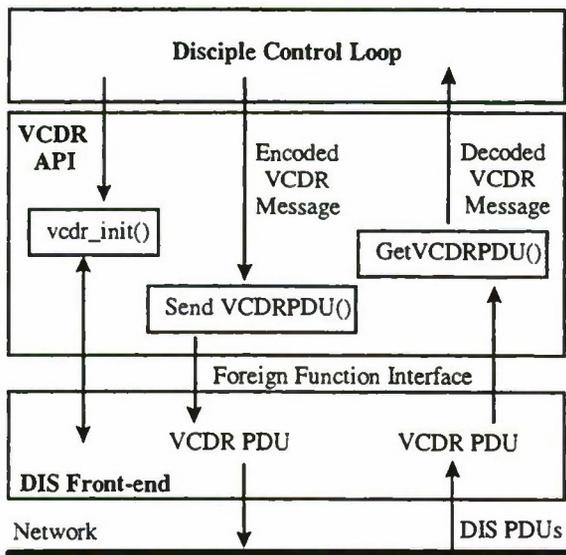


Figure 10: Disciple VCDR Components

4.2.2 Architecture

Figure 9 depicts the architecture of Virtual Commander consisting of ModSAF, Disciple, a new library for ModSAF called *libdisciple*, a DIS front-end for Disciple, and an API for Disciple.

The front-end developed for Disciple to send and receive DIS PDUs implements three API functions: *vcdr_init()*, *GetVCDR PDU()*, and *SendVCDR PDU()* as shown in Figure 10. The initialize function is called once when Disciple is started. It creates a socket with a well-known port to receive incoming messages. The *GetVCDR PDU()* function performs a blocking read of the socket. This is the state Disciple is usually in, waiting for the next request from the SME via ModSAF. Upon receiving a PDU the message contents are returned to Disciple. The *SendVCDR PDU()* function, on the other hand, packages and sends a message as a VCDR PDU to ModSAF.

In the newly created ModSAF library, *libdisciple*, several functions were added to perform the VCDR functions. These components and their interaction is shown in Figure 11. An editor was created, and is initialized in *discip_init_gui()*, for the GUI to accept commands from the SME. The learning data and phase of the learning cycle is packaged in the PDU and sent directly to Disciple's socket as a unicast message by *discip_done()*. (This is a departure from the use of ModSAF-provided functionality of broadcasting or multicasting.)

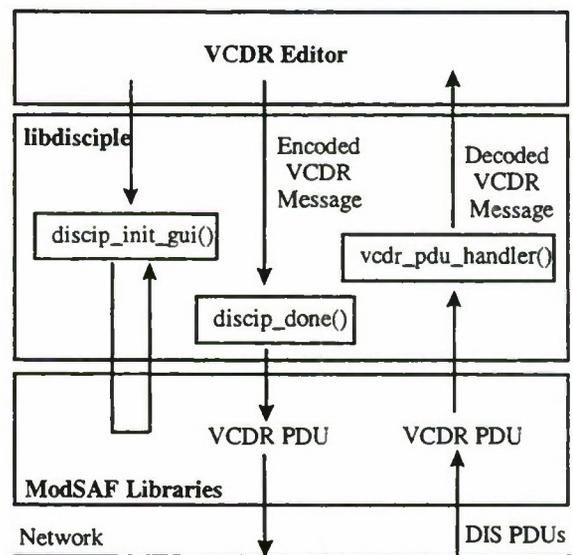


Figure 11: ModSAF VCDR Components

When the VCDR PDU is received by ModSAF it passes it to the callback function, *vcd_r_pdu_handler()*, that was registered earlier. From this point, one of several procedures is called based on the value of the phase field in the PDU. If appropriate, the plan view display or the Disciple editor in ModSAF will be updated to reflect the message in the last VCDR PDU received.

Although we have considered the fact that the messages between ModSAF and Disciple need to be delivered reliably, the current implementation does not guarantee that. It is common in situations like this for the application to take on the responsibility of ensuring successful transmissions where the network protocol is unreliable, such as with UDP. In the future, the use of a protocol such as the proposed Selectively Reliable Transport Protocol (SRTP) (Pullen and Laviano, 1995), which provides varying levels of reliability would allow the application to choose the degree of reliability necessary.

5. Related Work

With the advent of Command Forces (CFORs) and Intelligent Forces (IFORs) it has become necessary to add many more external reasoning and learning processes to their agent architectures. Some of these methods are already under development

MITRE is developing a Command and Control Simulation Interface Language (CCSIL) to provide a common language for CFORs (Salisbury et. al. 1995). CCSIL represents a language, a set of vocabulary and terminology, that is used to communicate between command entities and subordinate entities or vehicles. The interface is implemented by transporting the CCSIL messages within a DIS Signal PDU. This corresponds to the third approach that was described earlier. Additional effort is necessary to interface existing applications to DIS Simulations.

The Soar/IFOR research group also is developing an architecture to provide intelligent forces. The interface of Soar and ModSAF represents the first approach we described earlier. Soar is a general cognitive architecture that provides intelligent agents. The Soar architecture combines with the graphical interface, network interface, and scenario creation and execution tools of ModSAF to provide an architecture for intelligent forces. Currently, the

group is integrating the CCSIL command language (Hill 1996).

The RTI of the HLA presents another integration approach. The RTI will provide a set of services which facilitates integration of simulations, C⁴I systems, and engineering models. The RTI will utilize a more reliable transport mechanism than DIS and the RTI APIs will be more standardized, which will eliminate some of the disadvantages of a DIS interface, which lacks a standardized API.

6. Conclusions

The choice of interface approach may be determined partially by external factors (as with COMPASS, where a client library was provided that facilitated RPC calls). However, there is always latitude in some aspects of the implementation. The crucial factors that will determine the best choice are 1) Data Representation; 2) Distribution of Processes and Associated Communication Issues; and 3) Transport Mechanism. Consideration of the issues raised above will allow a more informed design when constructing the interface between a DIS application and an external process as demonstrated in the case studies above.

7. Acknowledgments

This research was conducted in the Center for Excellence in Command, Control, Communications and Intelligence and the Computer Science Department at George Mason University. Work on ModSAF-COMPASS was sponsored in part by the Defense Modeling and Simulation Office (DMSO) under DISA contract DCA100-91-C-0033. The authors thank Dr. Gheorghe Tecuci for his assistance with the Disciple Learning System and the COMPASS group at SAIC, Inc. for their cooperation and assistance.

8. References

- Birrell, A., and Nelson, B. (1984). "Implementing remote procedure calls," *ACM Transactions on Computer Systems*, Vol. 2, pp. 39-59.
- Calvin, J. and Weatherly, R. (1996). "An Introduction to the High Level Architecture (HLA) Runtime Infrastructure (RTI)," *14th Workshop on Standards for the Interoperability*

- of *Distributed Simulations*, paper 96-14-103, May 1996.
- Ceranowicz A. (1994). "ModSAF Capabilities," 4th *Conference on Computer Generated Forces and Behavior Representation*, May, Orlando, Florida.
- Cypher, A. (Ed.). (1993). "Watch What I Do: Programming by Demonstration," *MIT Press*, Cambridge, MA.
- Harlequin Group Limited, The (1994). *LispWorks User's Guide*.
- Hieb, M.R., Tecuci, G., Pullen, J.M., Ceranowicz, A., Hille, D. (1995). "A Methodology and Tool for Constructing Adaptive Command Agents for Computer Generated Forces," 5th *Conference on Computer Generated Forces and Behavioral Representation*.
- Hieb, M.R., Tecuci, G., Pullen, J.M. (1996). "Virtual Commander – An Instructable ModSAF Agent," 6th *Conference on Computer Generated Forces and Behavioral Representation*.
- Hill, Jr., R. (1996). *Intelligent Forces for Simulated Environments: Contractor's Progress Status & Management Report*, May, 1996 <URL: <http://www.isi.edu/soar/ifor/reports/monthly/may96.html>>
- IEEE Standard 1278.2 (1995). *Standard for Distributed Interactive Simulation – Communication Services and Profiles*, Institute of Electrical and Electronics Engineers, Inc.
- Laird, J. E., et al. (1995). "Simulated Intelligent Forces for Air: The SOAR/IFOR Project 1995," 5th *Conference on Computer Generated Forces and Behavioral Representation*.
- Loral Advanced Distributed Simulation (1995). *ModSAF 2.0 Developer's Kit*.
- Pullen, J.M., and Laviano, V.P. (1995). "A Selectively Reliable Transport Protocol for Distributed Interactive Simulation," 13th *DIS Workshop on Standards for the Interoperability of Distributed Simulations*, Paper 95-13-102.
- SAIC (1995), *Interface Control Document (ICD) for the COMPASS Architecture*, for NRaD.
- SAIC (1996), *Common Operational Modeling, Planning and Simulation Strategy*. <URL: <http://prw7.saic.com>>
- Salisbury, M.R., Booker, L.B., Seidel, D.W. and Dahmann, J.S. (1995). "Implementation of Command Forces (CFOR) Simulation," 5th *Conference on Computer Generated Forces and Behavioral Representation*.
- Soar/IFOR Research Group (1996). *Soar/IFOR Documentation*. <URL: <http://krusty.eecs.umich.edu/ifor>>
- Sun Microsystems. (1987). "XDR: External Data Representation Standard," *RFC 1014*, June, 1987.
- Tecuci, G. (1988). "DISCIPLINE: A Theory, Methodology and System for Learning Expert Knowledge," Ph.D. Thesis, University of Paris South.
- Tecuci, G., Hieb M.R., Hille D. & Pullen J.M.(1994). "Building Adaptive Autonomous Agents for Adversarial Domains," *Proceedings of the AAAI 94 Fall Symposium – Planning and Learning: On To Real Applications*.
- Yang, Z. and Duddy, K. (1996) "CORBA: A Platform for Distributed Object Computing," *Operating Systems Review*, Vol. 30, No. 2, pp. 4-31.

9. Authors' Biographies

Elizabeth White is an Assistant Professor at George Mason University (GMU). She received M.S. and Ph.D. degrees in Computer Science from the University of Maryland in 1990 and 1996, respectively, and a B.S. degree from the College of William and Mary in 1985.

Ken Frosch is a Graduate Research Assistant in the C³I Center at GMU. He is currently on leave of absence from Lockheed Martin Federal Systems where he has been a Systems Engineer since 1991. He received his B.S. in Computer Science and Mathematics from the University of Wisconsin - Madison in 1991.

Vince Laviano is a Graduate Research Assistant in the C³I Center at GMU. He received his B.S. in Computer Science from GMU in 1995. He has published papers in the area of distributed simulations.

Michael Hieb received a Ph.D. in Information Technology at GMU in 1996. Dr. Hieb is currently working for SAIC on the Multiple Reconfigurable C⁴I Interface (MRCI) project. He has published over 20 papers in the areas of learning agents, multistrategy learning and knowledge acquisition.

J. Mark Pullen is Associate Professor of Computer Science and member of the C³I Center at GMU. Dr. Pullen was with the Defense Advanced Research Projects Agency (DARPA) from 1986 to 1992, where he was Program Manager for Advanced Computing, Networking and Distributed Simulation.

Session 7a: Individual Combatant Behavior

Karr, UCF/IST

O'Keefe, U. S. Army Natick RD&E Center

Reece, UCF/IST

Fineberg, Pacific Sierra Research



Threat Analysis using Fuzzy Set Theory

Jaime E. Cisneros¹, Clark R. Karr¹, Dr. Pamela R. McCauley-Bell², and Sumeet Rajput¹

¹Institute for Simulation and Training
3280 Progress Dr., Orlando, FL 32826
ckarr@ist.ucf.edu

²Dept. of Industrial Engineering
University of Central Florida, Orlando, FL 32816
bell@iems.engr.ucf.edu

1. Abstract

One of the facets of realistically representing individual vehicles within Computer Generated Forces (CGF) systems is the targeting behavior of vehicles in the battlefield. Target analysis and selection involve several components, such as target detection and identification, as well as, threat analysis and selection. This paper describes a threat analysis algorithm based on Fuzzy Set Theory developed within the Modular Semi-Automated Forces (ModSAF) CGF system. Fuzzy Set Theory expresses ambiguous and complex situations, such as those that arise in real life threat analysis. IST's threat analysis algorithm produces, for each target, a membership value in the **Threatening** set by calculating and aggregating the membership values of the sets representing nine threat factors.

2. Threat Analysis

The target selection process is one of the many entity/vehicle behaviors that are challenging to simulate realistically. It consists of several facets: line of sight determination, target detection and identification, threat analysis, and target selection. This research concentrates on one facet of the targeting process: threat analysis. Threat analysis in CGF systems determines the threat level of an entity's detected targets. It features factors modeled after real life factors considered by soldiers in the battlefield. For the remainder of the report, "platform" means the entity doing the threat analysis, while target means the entity being analyzed.

2.1 Threat Analysis Background

Although threat analysis has complex and ambiguous situations in real life, it is simulated simplistically in CGF systems. Some of the better known CGF systems are: the IST CGF Testbed, ModSAF, and CCTT SAF.

2.1.1 The IST CGF Testbed

The IST CGF Testbed performs threat analysis by using a simple rule-based system encoded in C-code.

The simplicity of the rule-based system allows for ease of modification and maintenance. However, this simplicity doesn't account for many complex situations. The factors considered by this approach are:

- Vehicle type: Infantry Fighting Vehicle (IFV), main battle tank, etc.
- Target status: alive, damaged, flaming, or destroyed.
- Target actions: moving or stationary.
- Target type: IFV, main battle tank, etc.
- Vehicle's weapons' ranges.

All these factors, in rule form, are combined to obtain a threat value for a given target (Smith et. al. 1992a).

2.1.2 ModSAF

ModSAF's approach is more elaborate. It combines rules encoded in C-code and multipliers to scale a target's initial threat value. The initial threat value is based on the distance from the platform to the target and the target's acquisition level¹ (Loral 1994a).

2.1.3 CCTT SAF

From information that has been obtained, it appears that the CCTT SAF uses an approach similar to ModSAF's.

¹ ModSAF uses the NVEOL (Night Vision and Electronic Optical Laboratory) model to perform target acquisition. This model establishes that each sensor has a list of vehicles that are or could be detected. Each entry in the sensor's sensed list is coded by the acquisition level. Levels include: IN_FOV (target in sensor's field of view [FOV], but not detected), DETECTED (observer knows target has some military significance), CLASSIFIED (observer can distinguish general type, i.e. tracked vs. wheeled), RECOGNIZED (observer can distinguish function, i.e. APC vs. tank), IDENTIFIED (observer can distinguish model, i.e. T72 vs. T80).

2.2 Other Approaches To Perform Threat Analysis

There are other approaches that can be used to perform threat analysis such as Rule-based Systems, Neural Networks, and Fuzzy Set Theory.

2.2.1 Rule-based Systems

The IST CGF Tested and to some extent ModSAF use a simplistic rule-based system encoded in C-code. This approach is attractive because it is a natural way to express decision making, but these rules do not handle ambiguity well because they have to check hard boundaries. Rule-based systems can become complex and hard to maintain as factors and situations are analyzed in finer details (Charniak et. al. 1987). The analysis of finer details prompts the creation of more rules, increasing the complexity of the rule-based system. As additional factors are considered, the rule base tends to grow exponentially.

2.2.2 Neural Networks

Neural networks (NN) are typically organized in layers. Layers are made up of a number of interconnected nodes, which contain an activation function. Patterns are presented to the network via the input layer, which communicates to one or more hidden layers, where the actual processing is done via a system of weighted connections. The hidden layers, then, link to an output layer where the answer is produced (Hertz et. al. 1992).

Neural networks are in a sense the ultimate "black boxes." Apart from defining the network architecture, designing the training set, and seeding the interconnection weights with random numbers, the user has no other role than to input the training set, and wait for the NN to become trained (i.e. learn the training set). The learning itself progresses on its own. The final product is a trained network that provides no equations or coefficients defining a relationship (as in regression) beyond its own internal mathematics. The network is the final equation of the relationship. To this date, there is no obvious way to understand the meaning or even verify the correctness of the interconnection weights that come from the training.

2.2.3 Fuzzy Set Theory

Fuzzy Set Theory (FST) has provided a consistent and proven means to model many real world environments (King (1988) and Gupta (1988)).

FST does not sharply define sets as traditionally done in set theory. Set theory is governed by binary principles, such that a variable either belongs to a set (membership equals 1), or it does not belong to the set (membership equals 0). FST does not restrict set membership to complete (1) or none (0). Instead, it permits membership to be defined over the interval [0,1]. Membership expresses the degree an element belongs to a fuzzy set, and expresses the imprecision of many real world situations.

Membership functions are a characteristic of the data set under analysis, and can take on many forms. Several geometric mapping functions have been developed, including S, π , trapezoidal, triangular, and wedge shaped functions. Most of the membership functions that will be used in this experiment will be trapezoidal-shaped functions. The trapezoidal membership function is used to represent a set that is expected to exhibit a linear relationship. In this instance, there is not an optimal point or value which has complete membership. Rather, there is a range of values which have complete membership in the set.

The nature of FST makes it useful for handling a variety of imprecise or inexact cognitive conditions. "Inexactness" in cognitive information may arise due to a number of situations. To differentiate between these situations, or classes of problems associated with the use of FST, Kandel (King 1988) has subdivided the categories within which most problems assessed by FST fall. These include: generality, ambiguity, and vagueness.

1. *Generality* is the use of FST for specifying a general condition which can apply to a number of different states. A variety of situations can be characterized in this manner where the defined universe is not just a point (Gupta, Knopf, and Nikiforuk 1988).
2. *Ambiguity* is the use of FST to describe a condition where more than one distinguishable sub-concept can simultaneously exist.
3. *Vagueness* is the use of FST to present those cases whose precise boundaries are not well-defined. The boundaries in this case may be described as non-precise or non-crisp.

All of these types of fuzziness (i.e., generality, ambiguity, and vagueness) are present in real world applications, and can be represented mathematically

by a fuzzy set. This capability of FST prompted its use in IST's approach to threat analysis.

3. Threat Analysis using Fuzzy Set Theory

The perceived threat posed by various targets, in real life scenarios, is situation dependent, and a function of a variety of circumstances or factors, such as, emotional state, fatigue, attention, perceived operational activity of target, commander's intent, etc. For this project, IST's Subject Matter Expert (SME) isolated nine factors involved in threat analysis. These nine factors correspond to a mixture of high and low importance factors. The nine factors are:

1. Aggregate Threat Assessment.
2. Near Counter Threat.
3. Target's Effective Range.
4. Target Firing Status.
5. Aspect Angle.
6. Relative Elevation of Target.
7. Target Movement.
8. Target Type.
9. Sector of Fire.

3.1 Aggregate Threat Assessment

Ignoring other factors, a target in a threatening unit is more threatening than one in a non-threatening unit. For example, a target in an assaulting unit is more threatening than one in a unit doing a road march. The factors considered in the aggregate threat assessment are:

- Formation.
- Distance.
- Heading.
- Aiming status.
- Closing speed.
- Percentage of stationary targets.

The effect of the six unit factors is brought together to estimate the unit's level of threat. This is accomplished by performing an union operation of the factors, and is given by the equation:

Aggregate Threat Assessment =

$$\frac{\sum_{i=1}^6 \text{unit factor}_i}{\text{Number of unit factors}}$$

The value of each unit factor is in [0,1].

3.1.1 Formation

A visible target and the visible targets within a "unit radius" of the first target are considered to be a "unit." The position and spacing of the targets in the "unit" relative to the platform are analyzed to determine one of six formations: none, wedge, staggered line, column, line, and vee (Cisneros et al. 1995). Formation "none" has a value of 0; the others have a value of 1.0.

3.1.2 Distance

The distance from the platform to the target's unit has obvious importance; if the platform is within weapons' range of the unit, the target poses a threat. The weapons' range has been divided into 4 threat levels to indicate the threat level of the target's unit: minimal, marginal, lethal, and deadly.

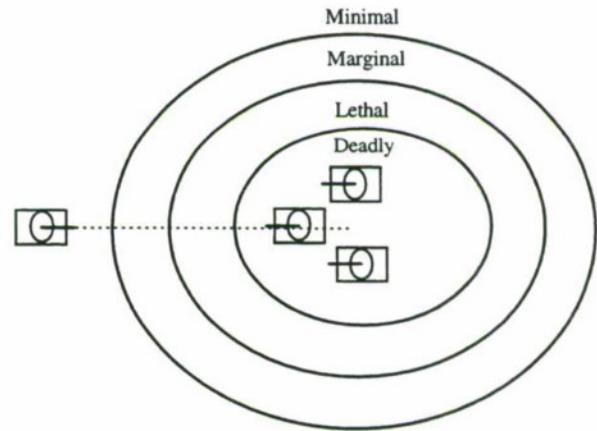


Figure 1: Lethality Areas.

The value of the distance factor is the degree of membership in (Figure 2).

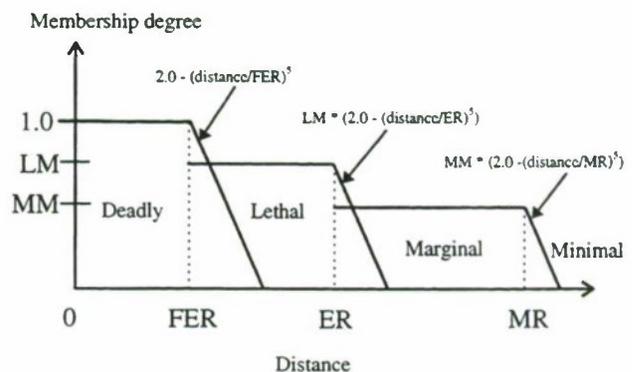


Figure 2: Weapon's Range of the Unit Fuzzy Set divided into four Fuzzy Sets.

where:

- FER: Fraction of the weapon's effectiveness range.
- EF: The weapon's effective range.
- MR: The weapon's maximum range.
- LM: The lethal area membership (parametric data).
- MM: The marginal area membership (parametric data).
- S: Exponent to select the steepness of the line (parametric data).

3.1.3 Heading

The target's unit direction of travel is used to estimate its threatening behavior. An enemy unit traveling towards a platform is considered more threatening than one moving away. A trapezoidal membership function centered on the line connecting the target and the platform with a parametric width supplies the membership value.

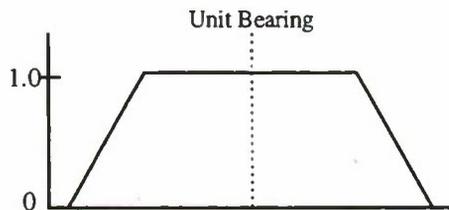


Figure 3: Membership function for Unit Heading

3.1.4 Aiming Status

The aiming status expresses the aiming behavior of the target's unit. A trapezoidal membership function centered on the line connecting the aggregate bearing of the guns in the target's unit and the platform with a parametric width supplies the membership value.

3.1.5 Closing Speed

The speed at which the unit is approaching the platform determines this factor's membership value. A wedge membership function centered on the target-platform line determines the threat value (Figure 4).

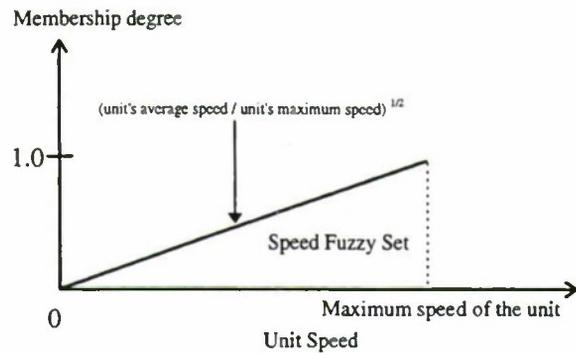


Figure 4: Speed Fuzzy Set.

3.1.6 Percentage of Stationary Targets

The percentage of stationary targets determines the number of unstabilized weapons that can be fired. The greater percentage of stationary targets, the greater the threat.

This completes the discussion of aggregate threat assessment. The remaining factors call for pairwise target-platform threat assessment.

3.2 Near Counter Threat

The threat level of a target decreases if there are friendly vehicles nearby that can destroy the target. This factor is modeled by:

$$\mu(x) = 1.0 - 0.1^{1/x}$$

where:

x: number of near counter threats.

Notice that as the number of nearby counter threats grows very large, the membership value does not go down to zero. Having many counter threats around does not mean that the target stops being a threat to the platform.

3.3 Target's Effective Range

A platform is in danger when it is within weapons' range of a target. To measure the threat level and model this factor, the weapons' maximum and effective ranges are used to create a set of ranges, corresponding to four threat levels: minimal, marginal, lethal, and deadly (see Section 3.1.2).

3.4 Target Firing Status

A target's firing behavior effects its threat level. A target that is engaging or preparing to engage the platform is quite threatening. A target that has fired at a friendly vehicle is less threatening, while a target that is scanning for vehicles is even less threatening. This factor considers four sub-factors, two crisp and two fuzzy factors, organized in decreasing threat order. The sub-factors are:

- Target has fired at the platform.
- Target is aiming at the platform.
- Target has simply fired.
- Target is scanning.

3.4.1 Target has fired at the platform

A target that has fired at the platform represents an enormous threat to the platform's survival. Direct fire at the platform, then, gives the highest membership value in the fuzzy set to the target. LibVEnemy (a ModSAF library) implements a task which accumulates incoming impacts near a vehicle, in order to determine whether the vehicle is under attack. Information about the number of rounds and the targets shooting those rounds is maintained. Query interface functions are provided to ask whether the platform **thinks** it is under attack, and if so, by whom (Loral 1994b). To avoid using ground truth, IST's threat analysis is performed only on targets that have been identified, that is, targets that can be found in the list of Spotted Targets. Therefore, determining if a target has fired at a platform can only be made if the platform "crew members" have seen the target fire at them. This is a crisp factor (the value is 0 or 1.0) represents no ambiguity about whether a target is attacking the platform.

3.4.2 Target is aiming at the platform

A target aiming at a platform is indicative of a target preparing to fire at the platform. Thus, a target aiming at a platform is more threatening than one that is aiming its main gun in some other direction. A trapezoidal membership function, based on the bearing of the gun relative to the platform, determines the value.

3.4.3 Target has simply fired

If a target has fired, it is presumably attacking friendly forces. This behavior increases the threat level of the target. LibVEnemy provides a way to

determine whether a local target has fired within a specified time. The limitation exists because fire PDUs are not monitored for non-local vehicles. To avoid misusing ground truth, only spotted targets are considered. The nature of the information provided by LibVEnemy is factual, so this factor is a crisp factor.

3.4.4 Target is scanning

A target demonstrates a threatening behavior if it is actively searching for vehicles to engage. Thus, a target that is estimated to be scanning for vehicles poses an obvious threat. This fuzzy factor is based on the degree of membership of the target's approximate scanning rate in the Scanning fuzzy set.

3.5 Aspect Angle

The aspect angle is the direction an aircraft is aiming its weapons. This factor is considered for aircraft only because aircraft don't have turrets or "main-guns" as ground vehicles do. The threat value of a target aircraft increases if it is lined up to perform an attack. To determine if a target aircraft is in position to carry out an attack, the bearing of the target aircraft is calculated. Then, the degree of membership is determined by a trapezoidal membership function:

3.6 Relative Elevation of Target

The relative elevation of a target is of importance because high ground gives an advantage to a target. This factor has less importance than others and was introduced to have a mixture of low and high importance factors, which is a reflection of real combat scenarios. The degree of membership in the Relative Elevation of Target fuzzy set is determined by a set of trapezoidal membership functions representing "below," "level," "above," and "definitely above."

3.7 Target Movement

This factor attempts to predict whether a target is becoming more threatening by moving closer to the platform. To make such prediction, the target's direction of movement is determined. Then, based on the direction of movement, the lethality area (see Figure 1) in which the target is attempting to move is estimated. The prediction is only good at the time of the analysis because this is a dynamic environment.

3.8 Target Type

The target type plays an obvious role in threat analysis. One way of organizing target types is by using an ordered list, as seen in ModSAF's Rules of Engagement Editor. However, this approach does not capture the fact that some target types are equally threatening. For example, a T80 may be as threatening as a BMP-2 carrying anti-tank missiles. IST's approach organizes targets types in three sets, corresponding to high, medium, low, and no priority sets. The target types in the same set share the same priority. This factor is modeled by giving full membership in the Threatening fuzzy set to targets in the High Priority Set, while giving partial membership to targets in the Medium and Low Priority Sets.

3.9 Sector of Fire

Typically, a unit commander assigns a portion of the battlefield to every vehicle in a unit. This portion of the battlefield is a vehicle's sector of fire. Vehicles use their sectors to scan and engage targets in an attempt to cover the unit's combat area. If a platform can estimate that it is within a target's sector of fire, the target's threat level increases because the platform will be engaged by the target.

The degree of membership within the Sector of Fire fuzzy set is determined by using a trapezoidal membership function with full membership for being in a pie-shaped area centered on the target's heading.

3.10 Determination of Threat Value

Fuzzy Set Theory provides a way to bring together the effects of different fuzzy sets. This operation is known as an aggregate operation on fuzzy sets. The aggregate operation that was selected to bring together the effect of all nine factors is:

$$\text{Threat Value} = \sum_{i=1}^9 \text{weight}_i * \text{factor}_i$$

Every factor_i has a weight_i associated with it. A weight_i is used to determine the relative significance of each factor_i within a mission, and to understand the interdependencies among the factors. Previous research has shown that this aggregation function is effective in simulation because it accounts for the contribution of every factor in the final result. In order to obtain the weights, Analytic Hierarchy Processing (AHP) (Saaty (1980)) was used to identify

the significance of every factor with respect to the others. This process starts by filling in a comparison matrix or AHP matrix, similar to the one given in Table 1 (F_i means Factor_{*i*}):

	F1	F2	F3	F4	F5	F6	F7	F8	F9
F1	1	3	1	5	5	5	3	5	5
F2		1	3	3	7	7	5	3	3
F3			1	5	5	5	3	5	5
F4				1	9	9	7	1	3
F5					1	1	3	9	3
F6						1	3	9	3
F7							1	9	3
F8								1	3
F9									1

Table 1: Comparison (AHP) matrix.

The meaning of the numeric values, the AHP ratings, assigned to each factor with respect to the others is explained below:

- Equally important: 1.
- Weakly more important: 3.
- Strongly more important: 5.
- Very strongly more important: 7.
- Absolutely more important: 9.

Once the AHP matrix is filled by a Subject Matter Expert (SME), the AHP Analysis is performed with the *Expert Choice Software* (Saaty 1980). This tool enhances the ability of the SME to consider how various alternatives of weighting would affect the outcome. This process produces weights for all the factors considered in the threat analysis (Table 2).

Factor	Weight
Aggregate threat assessment	0.079
Near counter threat	0.079
Target's effective range	0.158
Enemy firing status	0.301
Aspect angle	0.301
Relative elevation of target	0.021
Target movement	0.301
Target type	0.301
Sector of Fire	0.158

Table 2: Resulting weights.

The weights can be mission dependent, and they are specified prior to an exercise as parametric data.

4. Results

The threat analysis algorithm was implemented and tested in ModSAF version 1.5.1. Several scenarios were used to test the algorithm and the different factors. The scenario set included scenarios that were pathological to ModSAF's threat analysis. Based on SME evaluation, the FST threat analysis showed consistently realistic threat analysis. Cisneros (1995) describes a scenario used to test the threat analysis algorithm.

The platform was an M1A2 main battle tank. Four enemy targets and units were used:

- stationary BTR-80 facing the M1A2,
- retreating T72,
- a T72 platoon assaulting the M1A2 but starting much further than the above two vehicles, and
- a stationary T80 facing the M1A2 and located the farthest from the M1A2.

Even though the T72 platoon was farther than the BTR-80 and the retreating T72, it was considered the most threatening and was fired upon by the M1A2. After the platoon was destroyed, the farthest T80 was selected as the highest threat because it was stationary and aiming at the M1A2. Of the remaining capable targets in the scenario, the M1A2 picked the retreating T72 as the most threatening. After it was destroyed, the BTR-80 was selected and destroyed.

5. Conclusions

Threat analysis is an essential component of the targeting process. The Fuzzy Set Theory approach described in this report mirrors the human decision process, by taking into account the ambiguities and complexity of real life threat analysis. This approach performs threat analysis by considering nine factors, derived from information that was provided by a Subject Matter Expert (SME). This FST threat analysis approach provides an easily extendible and flexible mechanism for representing the complex threat analysis process in CGF systems.

There are several opportunities for future work within this area. It is possible to increase the realism of the threat analysis by:

- adding more factors,
- improving the analysis of some factors, e.g. unit formation,

- being able to change, dynamically, the relative importance of the factors, as changes take place in the battlefield, and
- using cooperative behavior and doctrinal tactics in the target selection process so that BLUEFOR and OPFOR forces perform differently in their targeting processes.

This research has shown FST to be a powerful technique for efficiently expressing ambiguous and complex battlefield situations. FST can be used in modeling many of the vehicle/unit behaviors, especially those that involve ambiguous and complex situation awareness.

6. References

- Charniak, E. and McDermott D. (1987). Introduction To Artificial Intelligence, Addison-Wesley Publishing Company, Inc., Reading, MA.
- Cisneros, J. E., Karr, C. R., and McCauley-Bell, P. (1995). "Intelligent Targeting in ModSAF," Contract report IST-CR-95-36, Institute for Simulation and Training, University of Central Florida.
- Gupta, M., Knopf, G., and Nikiforuk, P. (1988). Sinusoidal-based cognitive mapping functions, Fuzzy Logic in Knowledge-Based System, Decision and Control. Amsterdam: Elsevier Science Publishers, 69-92.
- Hertz, J., Krogh, A., Palmer, R.G. (1992). Introduction To The Theory Of Neural Computation, Addison-Wesley Publishing Company, Inc., Redwood City, CA.
- King, H. (1988). An expert adaptive fuzzy logic control system, Unpublished Doctoral Dissertation. Louisiana Tech University. pp. 1-57.
- Loral (1994a). "Libvassess documentation", Loral Advanced Distributed Simulation, Cambridge, Massachusetts, 1994.
- Loral (1994b). "LibVEnemy documentation", Loral Advanced Distributed Simulation, Cambridge, Massachusetts, 1994.
- Saaty, T. (1980). The Analytic Hierarchy Process, McGraw-Hill, New York, NY.

7. Acknowledgment

This research was sponsored by the US Army Simulation, Training, and Instrumentation Command as part of the Intelligent Simulated Forces project, contract N61339-92-C-0045. That support is gratefully acknowledged.

8. Authors' biographies

Jaime E. Cisneros was an Associate Computer Scientist in the Intelligent Simulated Forces project at the Institute for Simulation and Training. Mr. Cisneros has a Master of Science degree in Computer Science from the University of Central Florida. His research interests are in the areas of Natural Language Understanding, Machine Learning, and Computer Generated Forces.

Clark R. Karr is a Program Manager and the Principal Investigator of the Intelligent Simulated Forces project at the Institute for Simulation and Training. Mr. Karr has a Master of Science degree in Computer Science. His research interests are in the areas of Artificial Intelligence and Computer Generated Forces.

Dr. Pamela R. McCauley-Bell is an Assistant Professor in the Industrial Engineering department at the University of Central Florida. Her research interests are in Fuzzy Set Theory.

Sumeet Rajput is an Associate Computer Scientist in the Intelligent Simulated Forces project at the Institute for Simulation and Training. Mr. Rajput has a Master of Science degree in Computer Science from the University of Central Florida and is an MBA student at the University of Central Florida. His research interests are in the areas of Computational Geometry, Physical Modeling, and Computer Generated Forces.

Micro Resolution Terrain Processor (MRTP)

John A. O'Keefe IV
U.S. Army Soldier Systems Command
ATTN: SSCNC-AAM
Kansas Street
Natick, MA 01760-5015

Charles W. Howard, Ph.D.
and
Paul Saucier
Raytheon Company
Missile Systems Division
50 Apple Hill Drive
Tewksbury, MA 01876

1. Abstract

Micro Resolution Terrain Processor (MRTP) is computer program being developed by Raytheon Company under the US Army Soldier Systems Command's (SSCOM) Dismounted Infantry Support System (DISS) contract. This program is designed to be a terrain processor that can be incorporated into simulations such as the Integrated Unit Simulation System (IUSS) to provide extremely high resolution simulation of individual soldier visual search, target detection, and target tracking. It is designed to use 0.3 meter resolution terrain and feature data, plus simulate the effects of a full range of obscuration and battlefield illumination. MRTP has been designed to accept Defense Mapping Agency (DMA) formatted data for at least a 5 kilometer by 5 kilometer area and perform its calculations at faster than real time speed to support analytic Distributed Interactive Simulation (DIS) exercises.

2. Introduction

The US Army Soldier Systems Command's (SSCOM) mission is to develop, integrate, acquire and sustain soldier and related support systems in order to modernize, balance and improve the soldier's warfighting capabilities, performance and quality of life. SSCOM also performs similar functions for other services and customers. In developing equipment and clothing, SSCOM is taking a revolutionary approach to the oldest and most basic item of warfare by looking at the individual soldier as a complete weapons platform.

In order to assess the potential worth and contributions of proposed items on the soldier's performance and survivability, SSCOM has turned to

extremely high resolution modeling and simulation tools. These tools are applied early in the development cycle prior the construction of the first prototype. Initially, these tools use theoretical descriptions of proposed items in tools using first principle math and physics models combined in an integrated simulation architecture.

As prototype items are designed, built and tested the results of the early analytic simulations are reviewed and, where necessary, corrected and reanalyzed.

Recent use of simulation approach to support the design of the Force XXI Land Warrior has highlighted a requirement for a model that can recreate the individual soldier's visual search, target detection, and target tracking. Historical target detection models have demonstrated an inability to differentiate between the various individual vision devices and the soldier's unaided vision. This problem is further compounded by the fact that an individual human can be effectively obscured by terrain features as small as one foot in elevation, or horizontal width.

Members of the Soldier System user community have postulated that such a model must have the ability to use terrain databases having a 0.3 horizontal and vertical resolution.

MRTP was developed to meet these user and analytic needs.

3. Background

Analytic simulation of the incremental effects of addition/modification of individual soldier equipment on soldier performance and survivability requires

simulation of terrain at a level that replicates terrain and its uses by soldier in the real world. Terrain variations as small as those occurring in a 0.3 meter block of terrain will frequently have significant impact on the performance and survivability of individual soldiers. Therefore, terrain data and simulation at this level of fidelity are required to support analysis of proposed soldier system equipment.

The Integrated Unit Simulation System (IUSS) has been designed to model the effects of terrain, environment, load, battlefield challenges, and mission on the soldier's and small unit's mission performance and survivability. It uses available SIMNET Maneuver Control Console (MCC) or Janus format terrain data bases to accomplish this simulation and runs on Personal Computers (PCs) under Windows 3.11, Windows 95, and Windows NT operating systems.

The best currently available MCC or Janus format terrain data bases have a resolution of one meter grid spacing, the longitude and latitude distance between the elevation measurements. There are few readily available MCC or Janus format terrain data bases with this one meter resolution. One of these data bases, the Interservice/Industry Training Systems and Education Conference (IITSEC) terrain database of Ft Hunter Ligett, requires over 32 megabytes (MB) to store the 10 kilometer by 10 kilometer 1 meter resolution area. Increasing the resolution of this 10 kilometer by 10 kilometer area to 0.3 meter resolution would result in the 1.8 billion bytes of data just for the elevation data.

The addition of vegetation, roads, rivers, and other man made structures significantly increases the data that the computer must store and manipulate. For instance, during the development of the MRTP system software specification the number of trees in a 500 meter by 500 meter area located in Chelmsford, Massachusetts was surveyed. The area was selected due to its accessibility (it is the backyard of the Raytheon Company DISS Project Manager). The survey was accomplished by dividing the area into 10 meter squares and the trees within each square were counted. Approximately 11,400 trees of varying sizes were found in the surveyed area.

4. MRTP Functional Overview

MRTP has been designed to address the following general requirements.

- Load and use 0.3 meter Defense Mapping Agency (DMA) format terrain data bases,
- Load and use feature data bases,
- Operate at faster than real-time,
- Interoperate with IUSS, and
- Perform search, detection and tracking simulation for dismounted soldiers

MRTP has been designed to be a module that runs on a workstation connected to a network. It communicates with IUSS or other simulations connected to the network using UNIX socket communications protocols. The prototype provides the capability to provide simulation of the vision, search, tracking and target detection for a friendly and opposing dismounted infantry squad and seventeen other DIS entities. The interface to DIS is provided by IUSS or other DIS application.

Figure 1 provides an overview of a simulation's DETECT process. Initialization includes loading scenario data, processing command line arguments,

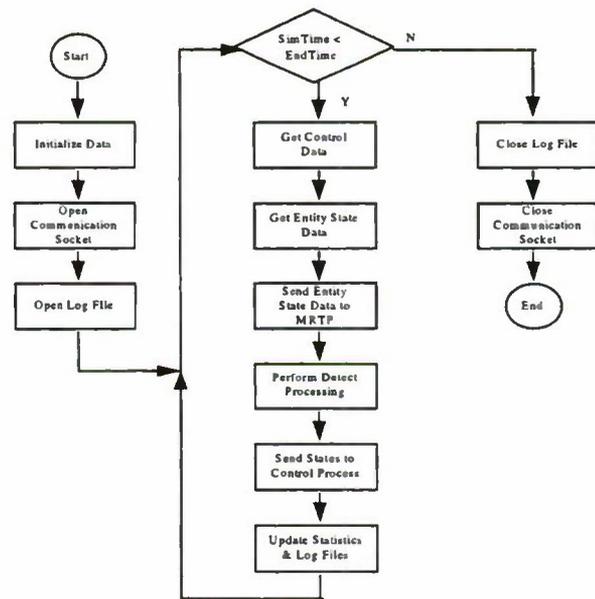


Figure 1: Host Simulation DETECT Process

opening log files, and establishing socket connections MRTP.

On each iteration of the main loop, the simulation time is advanced by 1 second, DETECT gets command, control and updated entity state data for all

simulated entities. DETECT sends a subset of the entity state data (time stamp, entity type, position, velocity, orientation) to MRTP. DETECT performs its own detection function while MRTP operates in parallel. At the end of each iteration, detection statistics are updated and results written to the output log files. When the detection function is finished, a status report is sent and the detection results are factored into subsequent command and control decisions, weapon assignment and engagement logic.

When the simulation time advances to the user specified scenario end time the simulation is finished and the main loop is exited, the log files and communication sockets are closed, and the DETECT process terminates.

Figure 2 illustrates a host simulation's detection function which is invoked from the DETECT main routine. As shown, this function iterates over the set of all simulated entities. If an entity is not "active" it is skipped and the next entity is processed.

For each "active" entity, detection processing is performed to determine if the sensors attached to the entity can detect the other simulated entities. While this is being performed MRTP does its own detection processing in parallel and sends the results to DETECT. Note, MRTP only performs detection processing for dismounted infantry soldier entities.

Figure 3 is an overview of the MRTP process. As shown, initialization includes reading a specification file which specifies the socket and host identification for MRTP to use when communicating with the DETECT process. It also specifies pathnames for the 0.3m resolution terrain database and the optional feature database. After reading the file, MRTP connects to the host simulation's DETECT process via a network protocol socket and then opens the terrain and (optional) features databases.

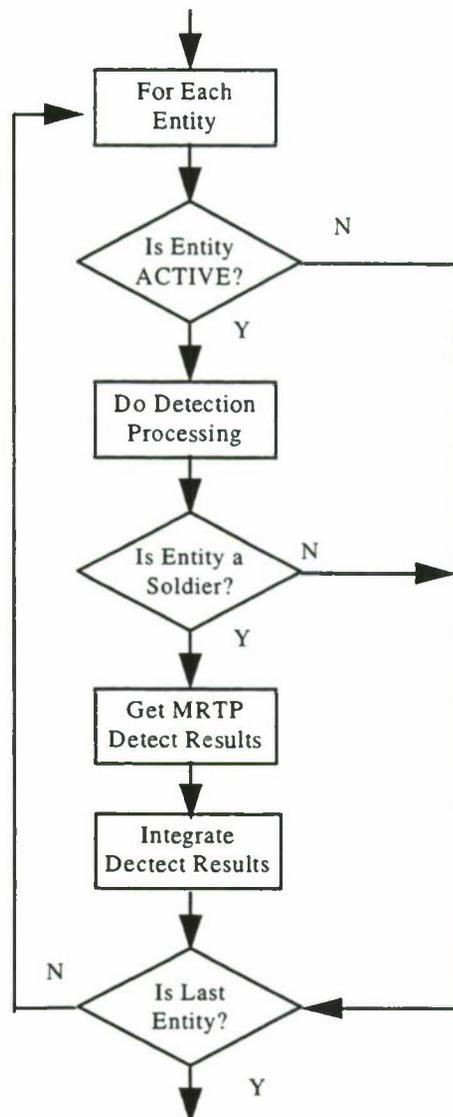


Figure 2: The Host Simulation's Detection Function

MRTP then enters its main processing loop where it gets current entity state information for each active simulated entity from the DETECT process. This occurs once for every simulation interval or 1 second.

The entity state data transmitted to MRTP at the start of each simulation time step is a stream of bytes of sufficient length to represent the state of all currently active simulated entities. First in the bytes stream is a header which is defined as follows.

```

typedef struct {
    PacketSize      Size;
    PacketType      Type;
} MRTP_PacketHeader;
  
```

The Type field contains the enumeration value EntityState=2 indicating the data is entity state information. The Size field specifies the number of bytes to follow and should be a multiple of size of(MRTP_EntityState). The rest of the byte stream is a sequence of bytes formatted as MRTP_EntityState structures which are defined as follows.

```
typedef struct {
    Entity_ID      ID;
    LONG           Kind;
    LONG           Time;
    XYZ_Vector     Velocity;
    XYZ_Vector     Position;
    XYZ_Vector     Orientation;
} MRTP_EntityState;
```

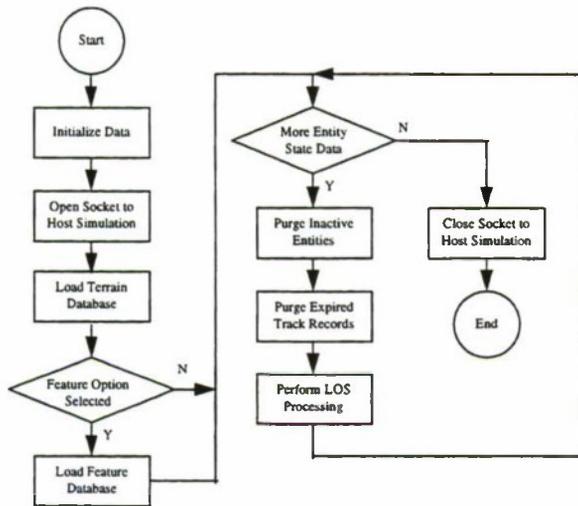


Figure 3: Overview of MRTP Process

ID is an identification number assigned by the host simulation system. It is assumed this number is unique and will not be reassigned to another entity or that the entity will not be assigned a different number during the simulation. It is not required that an entity be assigned the same identification number from one simulation run to another. The Kind field is a bit vector indicating whether the entity is a SOLDIER or NON-SOLDIER, FRIENDLY or HOSTILE. The Time field is the current simulation time. The Velocity, Position and Orientation fields are Earth Centered Inertial (ECI) vectors with double precision X, Y and Z coordinate values.

MRTP organizes entity state data into a linked list of MRTP_Entity structures. These are similar to the previous structures and are defined as follows.

```
typedef struct {
    Entity_ID      ID;
    LONG           Kind;
    LONG           Time;
    LONG           Step;
    XYZ_Vector     Velocity;
    XYZ_Vector     Position;
    XYZ_Vector     Orientation;
    DOUBLE         Rotation[9];
    struct MRTP_Track *TrackList;
    struct MRTP_Entity *Next;
} MRTP_Entity;
```

There are several new fields in this structure. The Step field counts how many time steps the entity has been active. This is not necessarily identical to the current simulation time since an entity does not have to be active at the beginning of a simulation. The Rotation[9] field is a double precision array used to rotate target vectors from ECI to body frame coordinates. The *Next field points to the next MRTP_Entity record in the list. The *TrackList field is a pointer to a (possibly NULL) list of track records. Track records model a soldier's ability to remember previously detected entities and are defined as follows.

```
typedef struct {
    Entity_ID      ID;
    LONG           LastSeen;
    struct MRP_Track *Next;
} MRTP_Track;
```

The ID field is the number of the entity being tracked. The LastSeen field is the simulation time when the entity was last detected. The Next field is a pointer to the next record in the list.

An important performance requirement of MRTP is being able to perform Line Of Sight (LOS) processing in less than 1 second of real time for each simulation time interval. This will enable MRTP to participate successfully in DIS.

M RTP is designed to handle a maximum of 18 dismounted infantry soldiers (in any combination of enemy or friendly) and up to 17 other entities, simultaneously. The collection of entities do not have to be identical from one simulation time step to the next. Entities may appear and disappear dynamically. Entities for which state data appeared in the previous step but not in the current step are “inactive” and are purged from the list while new entities are added to the list. Track records for targets that have not been seen for a specified period of time are also purged. These list maintenance actions are performed at the start of each simulated time step prior to performing LOS processing.

Figure 4 illustrates MR TP LOS processing. For each soldier entity, MR TP performs LOS processing with respect to every other entity. A non-soldier entity can only be a target viewed by a soldier in LOS processing. The non-soldier is never the viewing entity in MR TP. Of course, soldier entities don't attempt to detect themselves.

The first step in LOS processing is to compute the distance to the target. If the target is beyond the soldier's maximum viewing range it is classified as out of range and no further processing is done with respect to that target. If the target is within range, the target vector is computed and converted from ECI to body frame coordinates. This target vector is then used to compute the azimuth and elevation angles from the soldier to the target.

The next step is to select the soldier's field of view (FOV). This defines the viewing area in terms of minimum azimuth, maximum azimuth and elevation angles relative to the soldier. The FOV depends on target distance, the soldier's search pattern and whether the target is being tracked.

A narrow FOV (34 degrees or user input) is used to search for targets at distances of 25m to the maximum viewing range. The FOV shifts left and right over time through the 8 step search pattern depicted in Figure 5. The completed search pattern covers a total of 170 degrees and is repeated every 8 seconds.

A wide FOV ($3 \times 34 = 102$ degrees) is used to search for targets at less than 25m. The search pattern is the same as for the narrow FOV except the FOV is wider. The complete search pattern with the wide FOV covers a total of 238 degrees every 8 seconds. The rationale for the wide FOV is that entities at close range are hard to miss and can be seen in the viewer's peripheral vision. The wide FOV is also used for

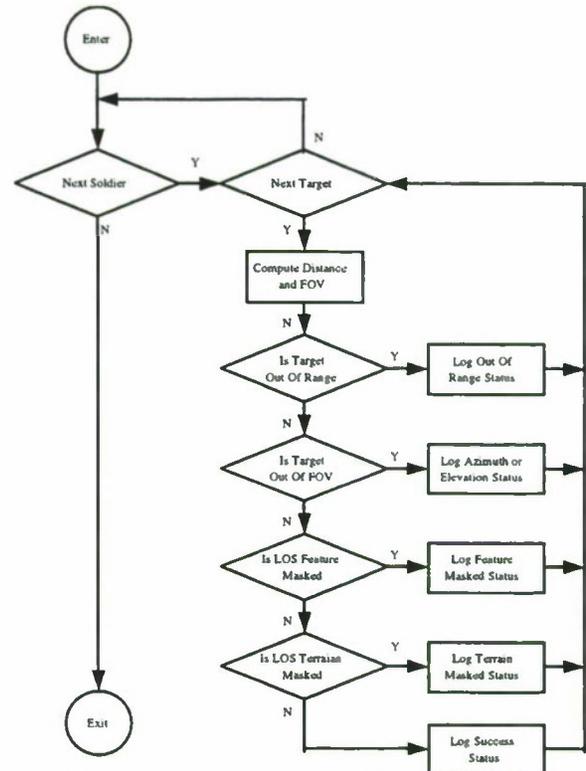


Figure 4: MR TP Line of Sight Processing

targets up to 100m distance that are being tracked. This reflects the fact that targets are more likely to be detected when the viewer knows where they are.

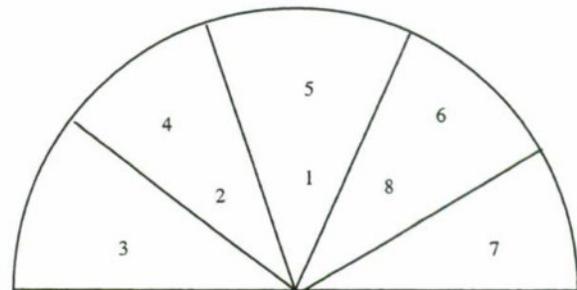


Figure 5: Field of View Scanning Pattern

If the range and FOV tests are passed then MR TP determines if the LOS is masked either by features or by the terrain itself. If the optional features database is present then MR TP determines if trees or other features block the LOS from soldier to target. If this test is passed, then MR TP goes on to determine if the terrain itself blocks the LOS.

To determine if a LOS is feature masked, the ECI positions of the two entity are first converted to latitude / longitude. The LOS from one entity to the other is then walked at approximately 0.3 meter

intervals. Whenever another 10m x 10m square is entered the linked list of trees in that square is traversed. For each tree, the point along the LOS closest to the tree is found and the terrain database consulted for the elevation of that point. If the elevation of the point on the LOS falls between the Max and Min elevations of the tree's cylinders and the distance from the tree to the LOS is less than the radius of the tree cylinder, the LOS is considered to be feature masked.

When LOS processing for a soldier is complete, MRTP sends detection status for that soldier to the host simulation. The detect status is a stream of bytes of sufficient length to represent all the current targets. As with entity state data, the detection status is preceded by a header in which the Type field contains the enumeration value DetectStatus=3 indicating the data is detection status data. The Size field specifies the number of bytes to follow and is a multiple of size of (MRTP_DetectStatus). The rest of the byte stream is a sequence of bytes formatted as MRTP_DetectStatus structures which are defined as follows.

```
typedef struct {
    Entity_ID      Source;
    Entity_ID      Target;
    LONG           Time;
    SHORT          Status;
} MRTP_DetectStatus;
```

5. MRTP Feature Database

At the present time the feature database only contains trees. Each tree is modeled as three stacked cylinders which are referred to as elevation bands. As shown in Figure 6, each elevation band has a height, radius and density attribute. The height is the height in meters from the base of the cylinder. The base of the lowest cylinder is sitting on the surface of the earth. The radius is the radius of the cylinder in meters. The density is not yet used but is intended to represent the density of foliage. It can be used to represent trees in different seasons and as a LOS masking probability factor when the LOS intersects the cylinder.

AUTHOR BIOGRAPHY

JOHN A. O'KEEFE IV, a graduate of Norwich University (BA 1975), American Technological University (MA 1981), the U.S. Army Infantry Officer Basic Course, the U.S. Army Infantry Mortar

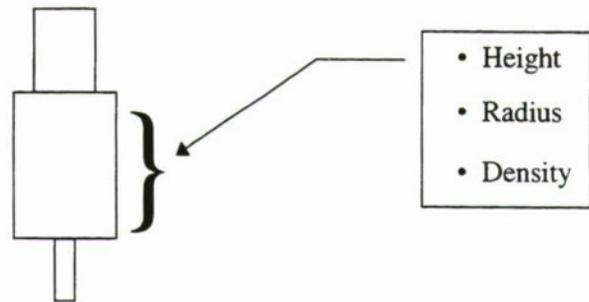


Figure 6: Current Tree Model

A simple template based algorithm is used to populate the database. It uses four template types; heavy (12 trees per 10m square), medium (10 trees per 10m square), light (4 trees per 10m square) or none (0 trees per 10m square). Although different size trees are supported in the design, the prototype implementation uses only a single type of tree.

The feature database is organized as a 500 x 500 grid overlaid on the 5km x 5km terrain. Each cell in the grid corresponds to a 10m x 10m area. Trees are planted in each square using one of the above templates. The database generator can be easily modified so trees are planted in any desired patterns.

The database has a 500 x 500 element header followed by an varying array of tree records. Each 500 x 500 header element corresponds to a 10m x 10m square and contains an index into the tree array to the first in a list of trees planted in that square. All the trees planted in the same square are linked together in a linked list.

6. Summary

A prototype terrain processor has been developed which supports simulation of individual soldier visual search, target detection and target tracking. This terrain processor is capable of supporting DIS applications such as IUSS using extremely high resolution terrain elevation and feature data.

Platoon Course, The U.S. Army Ordnance Officer Advanced Course, the U.S. Army Combined Arms Service Staff School, the U.S. Army Inspector General Course, the U.S. Army Command and General Staff Course, and the U.S. Army Materiel Acquisition Management Course, and a disable Regular Army Major, is a senior operations research analyst with the Advanced Concepts Division,

Advanced Systems Concepts Directorate, U.S. Army Natick Research, Development and Engineering Center. He is the project officer in charge of the development of the Integrated Unit Simulation System (IUSS) and the Soldier Protective Equipment Computer Aided Design (SPE CAD) system and is the Chairman of the Modeling Working Group of the U.S. Army Soldier System Technology Base Executive Steering Committee. He has been active in the application of modeling and simulation technologies to support materiel development since 1988.

Charles W. Howard is a senior analyst at the Raytheon Electronic Systems, System Design Laboratory. Dr. Howard has worked in research and design and development since 1978. Dr. Howard has managed many high level projects for the US Army in the areas of operations research and systems analysis, conducted numerous COEA's for the US Army and designed and managed these multi-team multi- year efforts for Air Defense, the future close combat vehicle man machine interface system. Dr. Howard has also managed major simulation models for Raytheon and the US Army during his career. Dr. Howard designed and managed multiple terrain and visualization workstations for the US Army Air Defense School during his employment at Fort Bliss as a Department of the Army civilian with the Directorate of Combat Development. His email address is charles=howard@sdl.msd.ray.com

Paul Saucier is a Senior Engineer at the Raytheon Electronic Systems, System Design Laboratory. Mr. Saucier has worked as a software engineer for over 18 years and has a MSCS degree from Worcester Polytechnic Institute.



Control of a CGF Fireteam with Voice and Gesture Commands

Douglas A. Reece
Institute for Simulation and Training
3280 Progress Dr., Orlando, FL 32826
dreece@ist.ucf.edu

1. Abstract

We have developed an experimental system that allows a human fireteam leader to direct a CGF fireteam using voice and gesture commands. This capability allows a small unit leader to be a *participant* in a simulation exercise rather than just an *operator* who controls CGF. This paper describes the interpretation of a simple set of voice commands and gestures by simulated individual soldiers. It briefly presents the recognition and encoding of commands into symbols. It next describes the architecture of the CGF system and the behaviors that implement the commands. The paper concludes by discussing the challenges posed in interpreting commands: understanding them in terms of the current situation and using common sense to fill in unstated information.

2. Introduction

We have developed an experimental system that allows a human fireteam leader to direct a computer generated forces (CGF) fireteam using voice and gesture commands. Such an interface to simulated entities addresses two shortcomings in existing CGF systems. First, command and control of CGF has generally focused on formatted orders, maps, and military symbols (e.g. Salisbury 1995). At the level of individual combatants, leaders do not generally collect radio reports and issue battle plans; instead they perceive the situation directly and give commands for immediate execution. Leaders point to objectives, locations, enemy forces, etc. that both the leader and his squad can see, rather than refer to named objects on a map. The language in which soldiers are trained uses spoken words and arm signals rather than text and graphical symbols on maps and overlays.

The second shortcoming of existing infantry CGF is that they are semi-automated: they must be controlled by a human operator with a computer keyboard and mouse (e.g. Franceschini 1994). The operator thus has an unnatural interface to the virtual

battlefield and cannot really participate in an exercise as a *trainee*. From the point of view of other trainees an opposing forces, the operator is an invisible leader who issues orders via an undetectable radio. The leader should be both visible and audible to friendly and enemy forces and vulnerable to destruction.

The experimental system constructed at IST consists of a leader interface component and a CGF component connected with a DIS interface. Gestures and sounds are encoded into symbols in the leader component. This paper describes how the encoded commands are interpreted by the CGF in the context of the current tactical situation. It does not describe the gesture or sound recognition in detail. After an overview of the entire system, the paper describes the CGF architecture, the command implementing behaviors, and the challenges posed in interpreting commands that may be ambiguous or incomplete.

3. System Overview

3.1 Requirements

The goal of this project was to build a prototype simulation system that would allow an individual in the role of a small unit leader to control a CGF squad and helicopter with gestures and spoken commands. The system was to use a limited set of standard military gestures. The voice commands were to use a natural set of sentences—avoiding any artificial use of coordinate systems or other artifacts of the CGF system—but use only a simple grammar to avoid the complexities of natural language understanding. Communication between the leader and the CGF system was to use the DIS (2.0.3) protocol as far as possible. By using this standard protocol, the system could function correctly if either the sender or receiver of commands were a trainee or CGF.

3.2 Command Set

The voice commands were composed from seven words using the following grammar:

```

START => "move" WHERE |
        "shoot" WHERE
WHERE => WHEN | "there" WHEN
WHEN => "now" | "when_he" COND
COND => "moves" | "shoots"

```

In the above production rules the capitalized words are symbols that must be expanded and the quoted words are terminals that are actually spoken ("when he" is treated as a single word). The phrases separated by "|" are alternate definitions of the name on the left. For example, grammar can produce the following phrases:

```

move now
shoot there when he shoots
move when he shoots

```

This grammar can produce 12 valid phrases.

Fourteen gestures are recognized the system. Thirteen of these are commands which are recognized and encoded as symbols, and one is simple an arm azimuth angle which is encoded as a numerical value. The arm azimuth is used in conjunction with voice commands and the Change Direction command to indicate locations, targets, etc. The command gestures include the following:

FORM_COLUMN	FORM_LINE
FORM_WEDGE	DISPERSE
OPEN_UP	CLOSE_UP
INCREASE_SPEED	DECREASE_SPEED
ADVANCE	HALT
CEASE_FIRE	COMMENCE_FIRE
CHANGE_DIRECTION	

Helicopter landing signals were also implemented, but they will not be discussed here.

3.3 System Architecture

Figure 1 shows the overall architecture of the system. The trainee station includes everything except the CGF component. A personal computer running DragonSpeak software accepts voice input and recognizes words. These are encoded and sent over a serial line to the workstation. Position information from three sensors (one on each hand and the back) is also sent to the workstation, where gestures are recognized from the data. The workstation hosts a DIS simulation which maintains the trainee's position in the virtual world (movement is directed with a joystick) and produces the visual output for a head mounted display. Encoded gestures and words are added to the DIS Entity State packet and broadcast on the network. Arm azimuth can also be added to the Entity State packet as an articulated part parameter.

The CGF system receives Entity State PDUs and extracts the gesture and voice information just as with all other state information. The CGF soldiers can observe the leader and see that he is making a gesture, and hear the leader's spoken words. Each CGF soldier must parse the spoken words to determine if the leader gave a valid command.

3.4 Gesture Recognition

Gestures are represented as a network of static poses. A pose is one hand's orientation and position relative to the body. Inputs to the network are "sensors" which detect when the hand moves to the position defining a pose. The network specifies sequences of poses, combinations of poses, and rates of pose changes that must be detected to recognize the gesture. As the hands are moved, all gesture networks are updated until one produces a

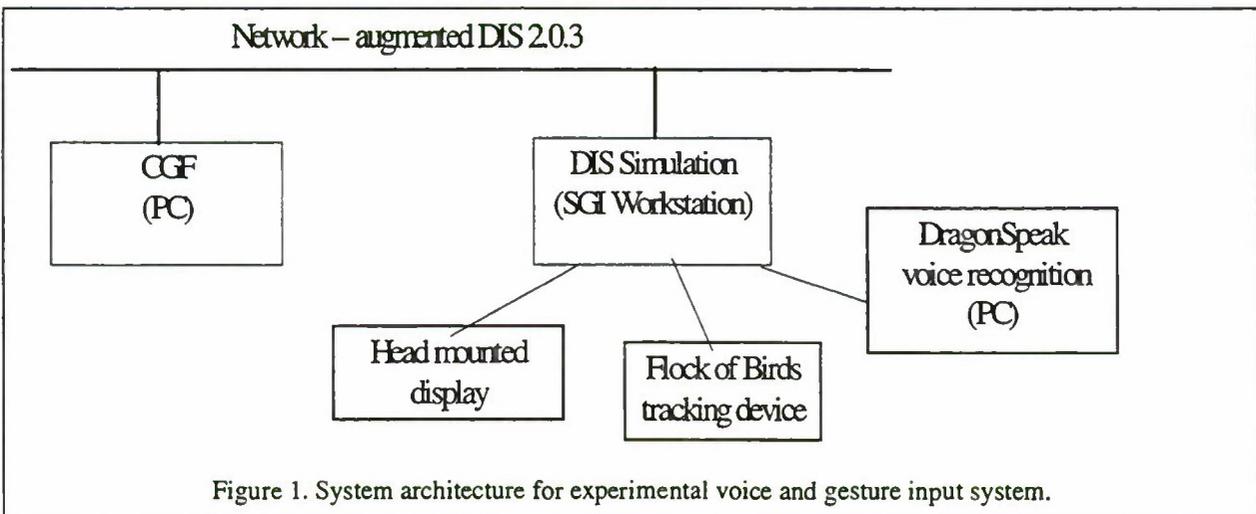


Figure 1. System architecture for experimental voice and gesture input system.

“recognized” output. Further detail on the gesture recognition system can be found in (Abel 1995).

4. The CGF System

4.1 Basic Functions

The CGF used in this experiment is based on the IST CGF system. This system is a complete DIS simulator node that covers functions from network interface to behavior generation. Basic functions in the CGF system include the following:

Distributed Simulation Support—interface to the terrain database, including derived representations of buildings; dead reckoning of ground truth for all entities; translation between internal and DIS representations of entities. For voice and gesture recognition, the representation of human entities was extended to include the gesture currently being made and the word currently being spoken.

World Physical Model—line of sight calculations; probability of kill lookups for munition impacts; and movement updates with terrain surface and obstacle collision constraints.

Entity Physical Model—sighting (detection) model; hearing model; probability of hit calculation; fuel or energy expenditure model; control of weapon firing; and feedback controlled movement. For this voice and gesture project we deliberately simplified the problem by not requiring CGF soldiers to be looking at the leader in order to see gestures; thus the physical model was not extended.

Internal World Model—maintenance of list of detected, visible, and previously detected but currently invisible entities; local map of movement obstacles.

Behavior—situation assessment and action selection; selection of precomputed cover locations; movement route planning; formation movement; target selection (based on threat, mission priority, and fire distribution); environment scanning. Behavior is generally encoded in hierarchies of tasks. Within each task, current internal and external conditions determine what subtask should be performed to accomplish the task. All levels of task are reexamined every second or so in order to respond to changing situations. The words and gestures of a commander are observable phenomena in the environment just as the terrain and other aspects of

entities are. As part of the periodic situation assessment, the CGF soldier considers what word is being spoken or gesture given. He also considers what commands he is currently obeying. These inputs, along with threats, terrain, etc., help determine what action is to be taken.

While the IST CGF system can act as a semi-automated force (SAF), for this and other projects the entities are completely autonomous. When an individual combatant entity is created it begins a general behavior which includes following a mission, responding to threats, etc.

4.2 Incorporating Leader Commands

While the general concept of initiating a CGF behavior in response to a command is simple, the actual implementation requires specific details of timing and information sharing to be considered. Such details include the initial information given to CGF entities, the timing of commands, and the use of pointing gestures with certain commands to provide command parameters.

The CGF entities must be initialized before commands are given. This initialization parallels the unit knowledge that a soldier would receive in training. The steps are as follows:

1. Designate commander for each soldier. Use DIS entity ID.
2. Assign soldiers to a role in a unit. The role determines their position in a formation.
3. Give the soldiers their mission, which causes them to respond to their commander, react to threats, etc.

DIS and computer simulations in general model continuous phenomena with periodic updates. This fact forces us to define rules for the timing of commands:

- Commands must be held for about a second to guarantee that they are recognized. This is because the CGF soldier only checks the commander’s appearance about once a second. For this project the trainee station continues to send out the last recognized command forever so this requirement is always met.
- “No command” must be used between repeated commands to separate them.

Several commands use parameters. When giving the “change formation direction” command, the commander must point to the direction of desired

movement. Voice commands also use pointing gestures, as follows:

- When speaking the first word, the commander must face soldier who is to receive command. If more than one soldier is aligned, then both will receive the command.
- When saying “there,” the commander must point to a location or target with his arm.
- When saying “when he,” the commander must point to the trigger entity, who must be friendly.

4.3 Parsing Voice Commands

A voice command has several components: a type, a target or location, a condition, and a trigger entity. The target and condition/trigger are optional. Figure 2 shows the states of the parser that interprets commands and fills in command components. This parser corresponds to the grammar given in Section Error! Reference source not found..

4.4 CGF Behavior

In order to understand what to do when after command is received, a CGF soldier must maintain several pieces of state information. These include the active commands, the subtask, the rules of

engagement, and the formation movement parameters.

We found that for the command set used in this project, it was necessary for the CGF soldier to remember two simultaneously active commands: a “movement” command and a “fire” command. For example, a soldier might be following “fire there when he fires,” and at the same time executing a number of different movement commands. New movement or fire command always supersede the current ones. For a more extensive command set, it might be necessary to remember more than two active commands, or a history of recent commands; in addition, the developing situation might determine which commands remained active. In the present case conditional commands expired when the conditions were met.

CGF soldiers start the exercise in a top level task that has them follow orders and respond to threats. They begin in a subtask called “Wait.” Commands and threats cause them to transition to other subtasks in which they move to cover, move in formation, move to a commanded position, or engage the enemy. The conditions that cause the transitions between these subtasks are shown in Figure 3..

A number of assumptions were made during the

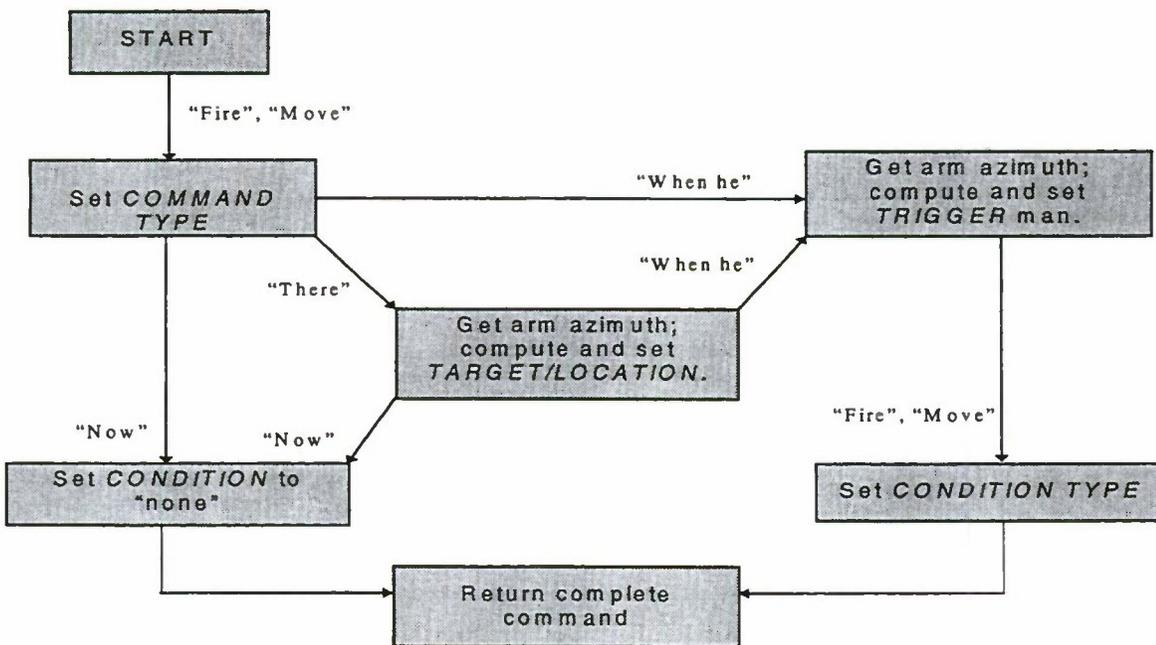


Figure 2. Voice parser states and actions. In all states, a “no word” input keeps the parser in the same state; a timeout or any word not listed returns the parser to the Start state.

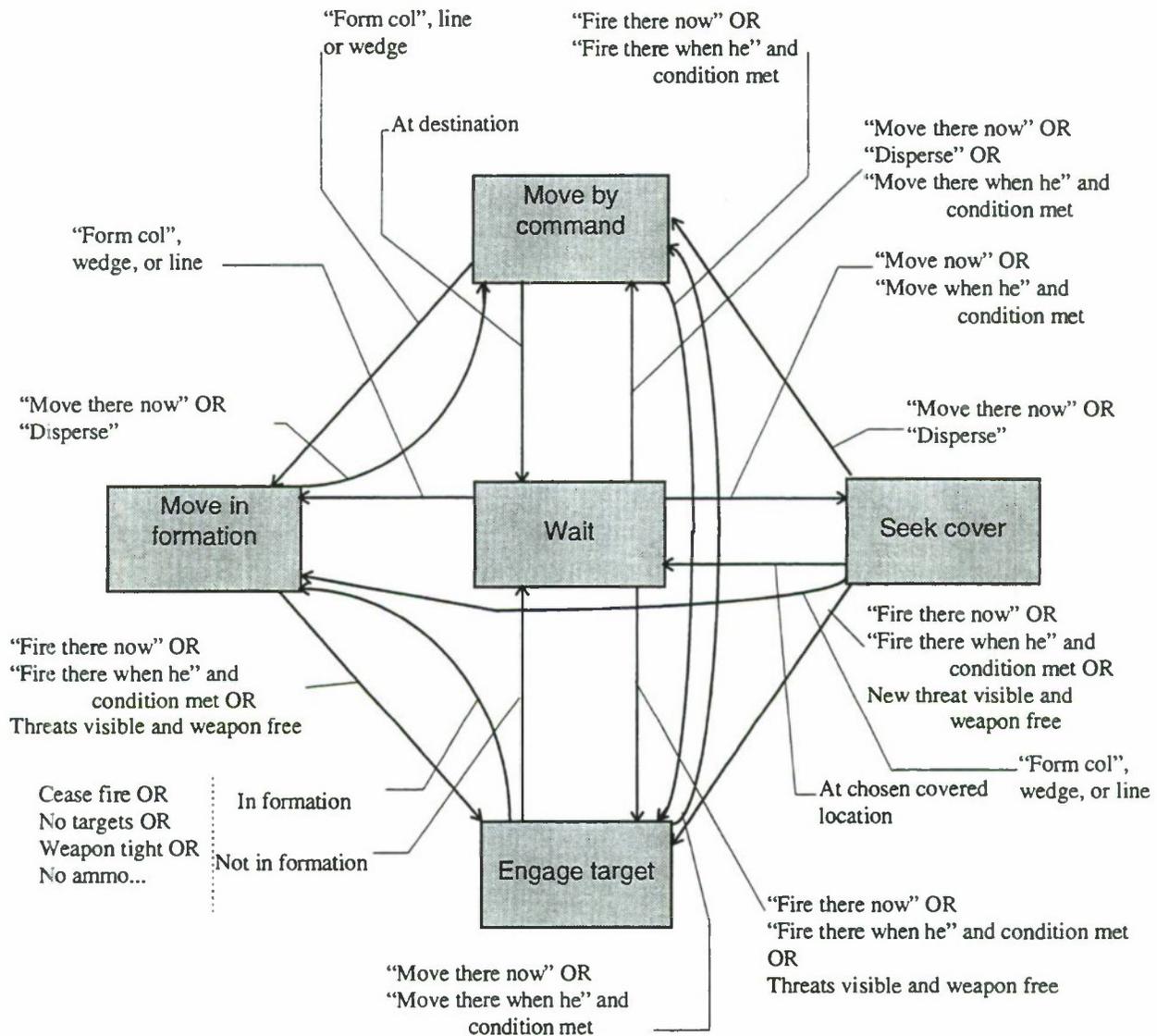


Figure 3. CGF Soldier subtasks. Commands containing "...when he..." are conditional; in this figure, "condition met" means that the "when he" condition has been satisfied.

design of the subtask selection conditions described in the figure. For example, when the CGF soldier is in formation movement, and there is an opportunity for it to fire at a threat, it is not obvious whether the soldier should stop formation movement and engage the threat. As the figure shows, we decided to allow a soldier in such a condition to engage the threat. On the other hand, if the soldier is moving in response to a "move..." voice command, it is not allowed to engage the target until the destination is reached.

The third important aspect of the CGF soldier's state is rules of engagement. Various commands and conditions change the rules from "tight" to "free"

(allowed to fire) and back. When the exercise starts, the soldier is in "weapon tight" state. "Commence fire," "fire now" and "fire there now" free the soldier to fire. "Fire there when he..." frees the soldier to fire when the condition is met. These commands free the weapon even if the soldier does not fire because there are no targets. The soldier only transitions to weapon-tight again if a cease fire command is given. Thus after the soldier is told to fire at one target, he may later engage other threats without a specific command if the opportunity presents itself.

The fourth piece of state information is the formation status. The CGF soldiers know whether they are

currently moving in formation, and know the current parameters of the formation including type (wedge, column, or line), scale (how spread out it is), and speed. These parameters are adjusted by commands. In order to move in formation, CGF soldiers also must remember information given to them about their unit. At the beginning of the exercise, a CGF operator must list the identity numbers of all soldiers in the fireteam. The operator also designates a role for each soldier in the fireteam. Each CGF soldier remembers this information..

A CGF soldier recognizes the following formation commands:

FORM_COLUMN	FORM_LINE
FORM_WEDGE	DISPERSE
OPEN_UP	CLOSE_UP
INCREASE_SPEED	DECREASE_SPEED
ADVANCE	HALT
CHANGE_DIRECTION	

The "Form_X" commands cause the CGF soldier to start formation movement activity. The other commands are ignored if the soldier is not in formation. Only a "Disperse" command or an individual movement command by voice breaks the soldier out of formation (although soldiers may temporarily break formation to engage threats).

One of the roles in the formation is always designated "point." Generally, the other fireteam members key their movement off of the point man. The "Change direction," "Advance," and "Halt" commands are only acted on directly by the point man; other fireteam members react to the point man's action.

4.5 Inferring Target Points

Several commands, especially voice commands, leave exact target locations or entities unspecified or indicated only with a pointing direction. The exact location or target entity must be inferred by the CGF soldier. This is done as follows:

MOVE THERE... The commander indicates "there" by pointing with his arm. If there is a feature providing cover (in our scenarios, a building) in the indicated direction and within 100 meters, the soldier moves to the building. Otherwise, the soldier moves to a point 20 meters from the commander in the indicated direction.

FIRE THERE.... The CGF soldier identifies the nearest hostile entity within 5 degrees of the angle in which the commander is pointing.

MOVE NOW. When no destination is indicated, the soldier finds the nearest covered location and moves there.

FIRE NOW. When no target is indicated, the CGF soldier fires at the highest priority target.

WHEN HE... The CGF soldier identifies the nearest friendly entity within 5 degrees of the angle in which the commander is pointing.

DISPERSE. When told to disperse, the soldier finds the direction directly away from the center of the fire team formation. If there is a covered position in this direction, that is chosen as the destination. Otherwise the soldier moves to a point 25 meters away from the fire team center.

5. Discussion

One of the important system lessons learned was that while commands may seem simple, their exact meaning in all circumstances is not obvious. For example, if a fireteam is moving by command and receives a fire command, can it stop and fire? If a soldier stops to fire, what do the other soldiers in the formation do? If a soldier receives a command to move here and then immediately after a command to move there, does it move here first and then there, or just ignore the first move command? If a fireteam moving in formation receives an order to change direction 180 degrees, how do the soldiers move to reorient the formation? Some of these questions can be answered with a little explanation from a subject matter expert; others require more extensive descriptions of behavior in different tactical situations; others require identifying and encoding common sense knowledge or common doctrine and incorporating it into the CGF soldier; and others require establishing constraints on the sequence of commands or the parameters for certain commands. In our behavior implementation we created active rule, subtask, engagement rule, and formation parameter states (that were not part of the gesture descriptions) to help define CGF soldier reactions to commands. In general system designers must be aware of the complexity of specifying command semantics.

The second lesson concerning the CGF system is that CGF soldiers must have adequate common sense, tactical skills, doctrinal and unit knowledge in order to execute commands properly. As mentioned above, doctrinal knowledge may be required to understand

the semantics of a command—e.g., how to change the formation direction. Tactical skills such as use of cover and concealment while moving, selection of a firing position, or direction of cover fire are necessary to act as a trained soldier. Common sense is necessary along with the tactical knowledge to understand the leader's intent and thus identify unstated or imprecisely indicated movement and fire targets. Unit knowledge is necessary so that the soldier knows who his commander is, who is in his unit, what his own role in the unit is, what the chain of command is, and who his parent unit is. Knowledge of the mission and unit tasks may be necessary to carry out orders according to the commander's intent. These requirements can lead to an extensive initialization process; for example, for a 10-man squad, each of 10 men must be told the identity and role of all 10 members of the squad—100 items of squad-role information. It is this body of knowledge that allows the squad to be controlled with a small set of simple commands.

6. Conclusions

Voice and gesture based communication are important for inserting individual combatants into virtual battlefields. We have describe an experimental system in which a small set of squad level command gestures and natural voice commands were used to control a CGF fireteam. While the commands all seemed simple, implementing realistic behaviors was challenging because the operational semantics of the commands had to be defined for the many command combinations and a variety of situations.

7. Acknowledgment

I would like to acknowledge the work of Kimberly Abel Parsons, who was the co-Principal Investigator on this project. Mike Smith, Kevin Meuller, and Paul Kelly implemented much of the software. The project was sponsored by the Army Research Institute under contract N61339-95-K-0001.

8. References

- Abel, K., Reece, D. and Smith, M. (1995) *Voice and Gesture Input for Computer Generated Forces: Final Report*, Institute for Simulation and Training, University of Central Florida.
- Franceschini, R. Petty, M. (1994) "Dismounted Infantry in DIS-Style Scenarios: A SAFDI Project Overview." In *Proceedings of the 4th Conference on Computer Generated Forces and Behavioral*

Representation. Institute for Simulation and Training.

Salisbury, M. (1995) "Command and Control Simulations Interface Language (CCSIL): Status Update, in *Proceedings of the 12th Workshop on Standards for the Interoperability of Distributed Simulations*, Institute for Simulation and Training.

9. Author's Biography

Douglas A. Reece is a Computer Scientist at the Institute for Simulation and Training. He is the Principal Investigator of the TTES Computer Controlled Hostiles project. His research interests are in artificial intelligence, specifically intelligent agent design and computer vision. He has a Ph.D. in Computer Science from Carnegie Mellon University and B.S. and M.S. degrees in Electrical Engineering from Case Western Reserve University.



Sensitizing Synthetic Forces to Suppression on the Virtual Battlefield

Michael L. Fineberg, Ph.D.
Pacific-Sierra Research Corporation
1400 Key Boulevard, Suite 700.
Arlington, VA 22209

Steven D. Peters
Micro Analysis and Design, Inc.
4900 Pearl East Circle, Suite 201E
Boulder CO 80301

Gene E. McClellan
Pacific-Sierra Research Corporation
1400 Key Boulevard, Suite 700
Arlington, VA 22209

1. Abstract

A major requirement for conducting effective command group training, tactical analyses, or weapon system evaluation within advanced distributed simulation (ADS) is the ability to exercise against realistic opposing forces (OPFORs). When soldiers are not available to command these opposing forces, we must rely on synthetic forces (SFs) to play their role. However, today's SFs do not always act with the variety and credibility of behavior associated with soldiers operating under the stress of battle. Consequently, the results of an exercise may not be completely valid for real world application. This situation is aggravated further by another fact: While the virtual environment and other conditions in an ADS exercise are becoming more and more representative of actual battlefield trauma, current SFs remain "insensitive" to these developing virtual battlefield conditions. This paper reviews a general theory of human performance under stress, efforts to use that theory as a basis for a taxonomy of human behavior, and an application of the theory and taxonomy to model the effects of suppressive fire on the behavior of virtual infantry teams.

2. Introduction

To further the integration of combatant behaviors into SFs and to make them more "sensitive" to simulated battle trauma, Objective 4 of the Department of Defense (DoD) Modeling and Simulation Master Plan (MSMP) requires the development of authoritative representations of the typical behaviors of combatants and teams engaged in hostilities and operations other than war (OOTW). This paper reports on the development of a unified theory of human behavior (UTB) and the behavioral taxonomy called out by Objective 4 of the MSMP. In addition, this paper describes an application of the model and taxonomy to the development of SFs that are sensitive to the effects of suppressive fires. As with other efforts described in the

MSMP, the overall purpose of this work was to enhance interoperability and reuse of the human behavior models and data developed to populate the behavioral taxonomy. In support of the objective of interoperability, the UTB allows the developer to "store" his human behavior model in its appropriate place relative to other contributions and to understand how his efforts relate to other models.

3. Toward a Unified Theory and Taxonomy of Behavior

The research team conducted a search of the literature on behavior theory, behavioral/task taxonomies, and the effects of suppressive fire on dismounted infantry that was published from 1960 through 1995. We identified 75 journal articles, books, and corporate reports and selected 60 studies for in-depth analysis, adding eight citations from other sources along the way. The *American Heritage Dictionary of the English Language*, 1992 edition, defines taxonomy as "the science, laws, or principles of classification; systematics." Within the behavioral sciences, the definition has been vague. DeGreene (1970) defined a taxonomy entry as a "verbal description using an object-verb format." In his view, taxonomies are essentially lists of verbs in hierarchical order that reduce behavior from higher level—observable and measurable behaviors—down to a fine level of "meaningless abstraction from the real world." Other authors (e.g., Levine 1971 and Meister 1985) have expanded the definition to include another idea: In formulating the structure of a taxonomy, it is crucial to understand and make explicit the model or theory of behavior underlying that structure. A taxonomy then, is not merely a list of labels with semantic definitions, it also must have syntactic structure.

The need to develop a unified behavioral theory and taxonomy that can be used across individuals and groups for multiple analytical purposes is a common thread that runs through

much of the research reviewed for this project. Such a taxonomy would allow behavioral scientists, computer scientists, engineers, trainers, and operational users to exchange data within a common framework and would eliminate the need to develop a new taxonomy for each new situation. Meister (1985) states that such a behavioral theory and taxonomy are required to analyze behavior to determine its constituent elements, to compare or relate two sets of tasks by their common underlying characteristics and behaviors, and to serve as the common basis for managing behavioral data.

3.1. The Theory

In response to this need, the first step in the construction of our taxonomy was to synthesize a unified theory of behavior (UTB) from past attempts to understand the relationships among the factors that underlie behavior under stress. The UTB shown in Figure 1 provides a context for the operation of the human behaviors to be classified and operationally defined in the taxonomy discussed later. It links the behaviors to their antecedents—through individual and team preparation for combat—and to their consequences in terms of battle performance measures.

The UTB is based on the work of several authors in the fields of human performance measurement

and stress research beginning with Cannon (1932) and Selye (1952, 1955, 1956) through the more recent efforts of Alluisi (1982), Lazarus and Folkman (1984), Gal (1985), Fineberg et. al. (1991), Conroy et. al. (1992), and Deitchman and Fineberg (1994 and 1995). The UTB bridges the gap between battlefield stress such as suppressive fire; the sensory, psychomotor, cognitive, social, and emotional responses to such trauma (stress symptoms); and the performance decrements as manifested in individual or team tasks associated with dismounted infantry operations and command and control (C²).

The UTB suggests that the antecedent conditions of generic battle stress (D1) and the specific combat tasks associated with particular scenarios (D2) interact to create an environmental demand on the individual soldier or team. Battle stress is influenced by variables such as combat intensity, weather, terrain, threat characteristics, force ratio, environmental toxins, wounds, and disease. The combat tasks to be accomplished influence demand through attributes such as number and duration of outputs required, continuous workload, difficulty of goal attainment, precision required, response rate, and procedural complexity (Fleishman and Quaintance 1984).

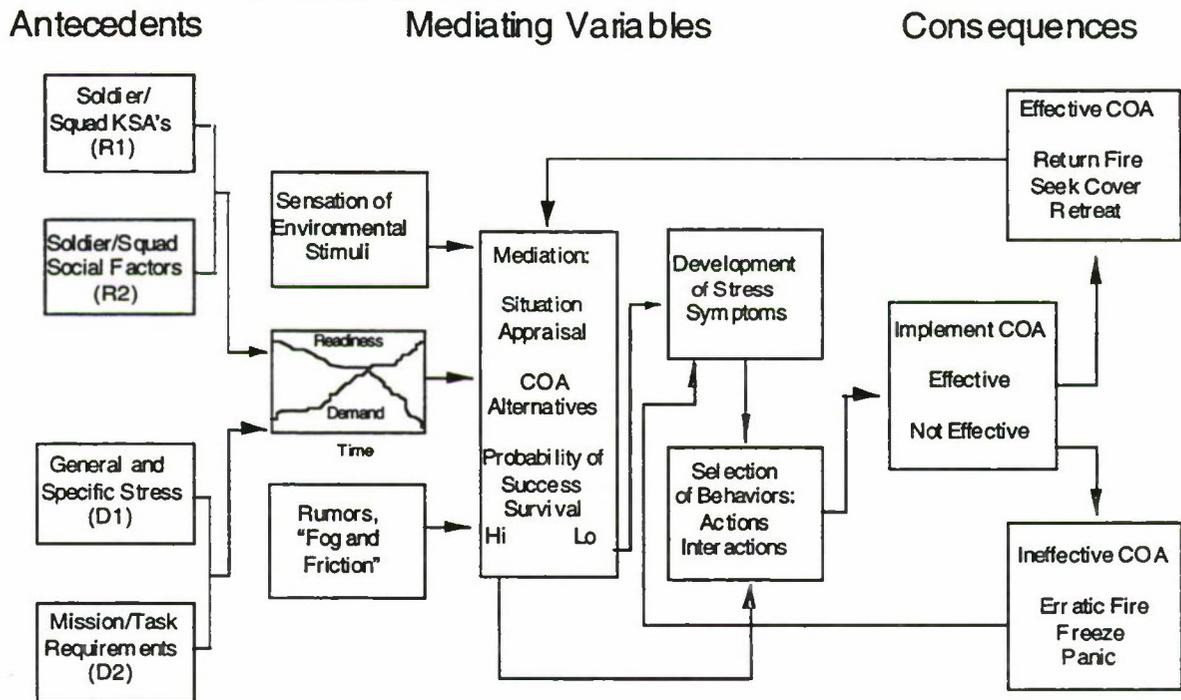


Figure 1: A Unified Theory of Combatant Behavior Under Stress

This resulting demand is met by another antecedent condition: the level of readiness to perform. Readiness is a complex function of an individual's knowledge, skills, abilities (KSA) and experience (R1) and interpersonal factors resident in his team and unit (R2). Abilities and traits include comprehension, expression, fluency of ideas, originality, memory, problem sensitivity, math reasoning, induction and deduction, and flexibility (Fleishman and Quaintance 1984). Interpersonal factors resident in the team include leadership, cohesion (horizontal bonding among personnel), commitment (vertical bonding to some ideal), role in the organization, and personal well-being (Blades 1986).

Demand and readiness are compared (either consciously or unconsciously) and the result yields an initial estimate of one's own ability to meet the perceived demand. For example, if readiness greatly exceeds demand, no negative effects on performance are perceived. As time passes, readiness may degrade, while demand may remain steady or grow. In any case, when the demand begins to approach the remaining readiness level, individuals use reserve capacity to meet the demand as predicted by the General Adaptation Syndrome (Selye 1956). At this stage, additional demands may cause the system to break down. The size of the readiness deficit and the individual's predisposition determine the character, prevalence, and magnitude of the performance decrement. This perception or estimate of one's ability to meet further demands, together with knowledge of the tactical situation on the ground and prevailing rumors and confusion, feeds into the mediation process in which appraisal of the situation and formulation of a course of action occur.

During mediation, the soldier forms a subjective estimate of his chances of survival and success. This estimate is based not on the situation per se, but on what the combatant tells himself about that situation. One is tempted to conclude that a low probability of perceived success will inhibit behavior, but risk-seeking behavior cannot be discounted. Perceived probability of survival and success, combined with adherence to appropriate tactics and doctrine, influence the selection and implementation of effective behaviors. If the perceived probability of survival and success is high, the soldier or team will use its training to select and employ the most appropriate behaviors in a course of action designed to resolve the situation. If this probability is perceived as low, selection of

behaviors will be "detoured" through a set of stress symptoms that tend to reduce the appropriateness and effectiveness of the behaviors and interactions selected relative to the task at hand.

The effect of the decrements in behavior modifies the initial capability of the individual or team by some percentage, leaving a residual capacity to perform. This residual capacity, as influenced by stress responses, translates to performances that are adaptive (advance or retreat) so long as they are not overly influenced by the stresses that drive them. Performance becomes maladaptive (panic and decisional paralysis) when the stress level exceeds some internal, idiosyncratic threshold.

3.2. The Taxonomy

The taxonomy is described in detail in Fineberg (1995). It is based on the behavior description and requirements approach described by Fleishman and Quaintance (1984) and on an information processing paradigm. The taxonomy contains four major taxons or classes of behaviors labeled Sensation, Mediation, Reaction, and Interaction. These major classes are analyzed further into 11 lower level categories that are populated by 188 action verbs. These action verbs are seen by the authors as building blocks of human behavior.

Taxon A, Sensation, contains the first two sub classes, that is, automatic and volitional behaviors that serve to collect, filter, condition, and retain data from the outside world for short periods. These data are passed on to Taxon B, Mediation, whose behaviors are sub-divided into three categories; preparing information for assessment, solving problems, and making decisions. Taxon B also includes the capability to revise decisions based on of knowledge of results. Taxon C, Reaction, implements the selected course of action by way of three additional categories of behavior; physical, psychomotor, and conceptual responses. It also keeps track of the results of these responses relative to task accomplishment. The behaviors in Taxon D, Interaction, communicate, coordinate, and advocate the selected course of action to superiors and implement this course of action by accessing three sub-categories of behavior designated; controlling, organizing, and leading. After developing the taxonomy, Fineberg (1995) constructed operational definitions for each behavior that specify the

meaning of a behavior by denoting its measuring operations.

4. Applying the UTB to Predicting the Effects of Suppressive Fires

Dupuy (1987) observed that “There is probably no obscurity of combat requiring clarification and understanding more urgently than that of suppression.” This is supported by battlefield data and anecdotal reports suggesting that the attrition aspect (casualties typically < 10%) of combat power alone does not determine battle outcome (Hughes 1995). Rather, the psychological and spiritual effects of combat power, the most measurable of which is suppression, will often have a far greater effect.

The phenomenon of suppression has been most recently defined (Hughes 1995) as “a non-lethal decrement in enemy combat performance from firepower that disappears when the battle is over.” Similarly, Dupuy (1984) saw suppression as the effect of fire on the behavior of hostile personnel, reducing, limiting, or inhibiting their performance of combat duties. The US Army Field Artillery School (1980) viewed suppression as the temporary degradation in combat effectiveness due to the terminal effects of explosive munitions. Finally, JCS PUB-1 defines suppression as a temporary or transient degradation of the performance of a weapons system below the level needed to fulfill its mission objectives. Rarely has there been so

much agreement on the definition of a psychological construct. This agreement, together with the acknowledged importance of suppression, reinforced our choice of suppression as the object of our current modeling effort.

4.1. Results from the state-of-the-art review

The effects of suppression have long been noted. Sun Tsu implied that the best battle is the one you don’t fight. Napoleon asserted that “the moral is to the physical as three is to one.” Clausewitz likened battle to walking upstream, that is, “action in war is like movement in a resistant element.” An unnamed Union general, when told that his artillery bombardment was off target, was quoted as saying, “Damn the effect! It’s the sound I want!” Throughout the history of warfare, the greatest military tacticians have been victorious by domination and control, not bloodshed and destruction. Perhaps the “smartest” weapons and tactics are those that most disrupt the morale and effectiveness of the enemy. If we can quantify the relationship between the parameters of suppressive fires and the behaviors of the troops at “ground zero,” suppression indeed may be “a good bargain at the price” (USAFAS 1980).

Our first step in quantifying those relationships was to characterize and operationally define the phenomenon of suppressive fire (see Table 1).

Table 1: Sources and Signatures of Suppressive Fire

Suppressive Fire Sources	Suppressive Fire Signatures	
Direct Fire Weapons Small arms Machine guns Tank mounted Anti tank weapons Characteristics Small area effect Hit or miss Point accuracy Lower fear factor Indirect Fire Weapons Artillery Air Strike Characteristics Wide area effect Collateral damage Higher fear factor	<u>Stimuli</u> Visual: Flash Smoke Debris Wounds Aural: Bang Whine Whiz Ricochet Screams Tactile: Heat Pressure Debris Wind	<u>Variables</u> Duration Magnitude Number Frequency Proximity Uncertainty Pattern

The stimuli and associated variables of suppression manifest themselves in changes in the behavior of combatants. These changes are mediated by constructs including radius of effect, troop deployment and posture, intensity of bombardment, and several miscellaneous variables (USAFAS 1980). Radius of effect can be expressed as: $R_m = 69.3(W_{hekg})^{1/2}$ where R is the radius in meters, 69.3 is a constant determined by a least squares regression, and W is the weight of high explosive in kilograms. This equation can be used to calculate equivalent areas for a given level of suppression. The area in which half of those under fire will not return fire ($P_s=.5$) is 2160 m² for 5 rounds of 50-caliber shells, 35,300 m² for one 155-mm shell, and 211,800 m² for one 8-inch shell.

With respect to deployment and posture on area of suppression, we find that troops in the open are suppressed by an artillery shell landing within a circle of 1000 to 2000 m² around their position. Troops in emplacements are not affected by suppression unless a shell lands within a circle of 300 to 500 m², and troops in armored personnel carriers (APCs) are not suppressed until the fire is within a circle of 120 to 140 m². Suppression also can be measured in terms of simple proximity of the burst to one's own position. Table 2 shows proximity in meters for several types of weapons relative to a desired level of suppression.

Table 2: Proximity Necessary to Result in Two Levels of Suppression

Weapon \ P _{supp}	.5	.9
M-16	1-3	
M-2	24-26	5-8
105 How	51-118	21-55
155 How	104-144	63-77
8 in How	257-392	126-169

Another predictor of suppression is the volume or intensity of fire. Table 3 indicates the number of shells per minute for various weapons necessary to achieve suppression at two probabilities.

Table 3: Rounds per Minute for Two Levels of Suppression

Weapon \ P _{supp}	.5	.9
M-16	88-128	293-413
M-2	23-25	75-100
105 How	5-10	15-25
155 How	4-10	12-25
8 in How	2-5	5-10

The delay in return fire resulting from suppression lasts either about 10 seconds or from 30 to 100 seconds, depending on whose data one accepts. Similarly a defender's return fire appears to be reduced by 80 to 90 percent for 15 to 30 minutes after heavy artillery bombardment. M-60 machine gun fire results in a 61 percent increase in tracking time of ATM gunners, and an interval of 4 seconds between M-60 bursts appears to cause the most suppressive effect.(USAFAS 1980)

Many other intervening variables have been noted in the studies we reviewed. Random distribution of fire throughout the target area is more suppressive than systematic patterns of fire. Those who are knowledgeable about the lethality of weapons are 40% more suppressed than those who are not. Soldiers operating alone are from 43% to 115% more suppressed than those who are with others. Those in a frontal parapet foxhole are 62% less suppressed than those in a conventional foxhole. The most suppressive fires occurred directly in front of the soldier, the least suppressive occurs directly behind him (USAFAS 1980).

4.2. Constructing a suppression-sensitive dismounted infantry(DI) team

Using the UTB to derive mathematical relationships and the parameters of suppressive fires from the literature as source terms, we began the process of representing the psychological phenomenon of suppression in Modular Semi-Automated Forces (ModSAF). We chose ModSAF 2.1 as the developmental tool for suppression-sensitive DI because of its capability

to create and control unmanned entities in the Distributed Interactive Simulation (DIS) environment. The suppression-sensitive infantry team is being constructed within a version of ModSAF 2.1 modified with indirect fire routines. This version was developed by the Leathernet support group at NRaD. Through the use of the DIS network, combat scenarios that include fire suppression effects will be played out and analyzed. The DIS environment allows manned and unmanned simulators to interact on a virtual battlefield through a network of computers. Our intention here is to describe the underlying principles that we used to implement suppression-sensitive performance into ModSAF, specifically for DI entities.

Computerized battlefield simulation exercises typically consist of combatants operated and controlled by manned and unmanned systems. To recreate accurate and realistic scenarios within the computer-generated environment, algorithms are required that model the effects present in live and actual battle situations. Extensive time and effort are spent on modeling the weapon characteristics and their performance capabilities. Military tactics and doctrine also are modeled to determine autonomously what actions will take place under various conditions that may be presented to the entity. Modeling how the human performs, however, has proven to be quite difficult.

4.2.1 ModSAF Characteristics

ModSAF contains synthetic entities designed to appear as though they are being maneuvered by human crews rather than computers. These entities can interact with each other and with manned individual entity simulators to support training, combat development experiments, and test or evaluation studies. Improving how each entity emulates the performance characteristics of its simulated human operator results in improved realism added to ModSAF. This improved realism results in an improved training capability. As implied in the UTB, ModSAF's computer generated forces will be modeled in a manner consistent with humans performing the identical task. DIs that encounter suppressive fire, while incorporating proper doctrine and tactics, will emulate appropriate (not necessarily perfect) human performance. That is, their selection of which tasks to perform next and their response time and accuracy in performing those tasks will be degraded in a manner consistent with known effects of suppressive fire on real troops.

Implementing suppressive fire phenomenology into ModSAF requires an understanding of its architecture. Today's ModSAF models all weapons, vehicles, and DIs as entities. That is, a common library of constructs is used to model a dismounted soldier and an M1 tank. Each entity has a set of parameters that describe its structure and they each use the same set of algorithms to describe their performance. In other words, both the tank and the DI use the same software routines to "know" how fast they can travel and how to follow a road. Although not all entities use and follow all the same algorithms, the current architecture does allow several different types to exhibit the same behaviors. Thus, modification of a behavior for one entity will affect other entities using that behavior.

Therefore, similar to an M1 tank, a ModSAF DI can move and turn in place, and each virtual soldier can carry and fire a weapon whenever a line-of-sight exists. Each DI can assume three postures: standing (in place or moving), kneeling, or prone with orientation determined by the terrain under him. A DI can mount appropriate vehicles (such as IFVs), ride to another location, and dismount. While mounted, DIs are not visible. In ModSAF 2.1, DI teams can consist of two or more individual DI entities. One member of a team may be configured with an anti-tank (AT) missile that can fire at tanks or aircraft (STINGER, SA-16). In addition, teams can move and/or "keep station" with each other. Indirect fire simulation currently includes the proper orientation of the gun and/or vehicle for particular fire missions.

When the ModSAF DI makes contact with an enemy entity, it executes a unit level reactive task that monitors enemy activity and reacts to the contact. A DI constantly checks to see if enemy entities are spotted. If so, the ModSAF DI reacts by executing an appropriate set of responses. The response sets that are currently supported are: Contact Drill, Assault, Withdraw, Occupy Position, and No Action.

If Contact Drill is chosen, the unit continues to execute its primary task while shooting at the enemy. If Assault is executed, the ModSAF DI creates an assault objective at the computed enemy location. If Withdraw is executed, ModSAF creates a point far from the enemy in the opposite direction where the unit can go. If Occupy Position is executed, ModSAF creates an objective facing the enemy and chooses logical target reference points with the

engagement area at the enemy location. All of these programmed sets of behaviors are predicated on the nature of the contact with the enemy and the predispositions input at the beginning of the engagement. They are not now subject to any intermediate influence of simulated battlefield trauma.

4.2.2 Developing the Suppression Sensitive ModSAF Dismounted Infantry (DI) Team

We used a proof-of-concept approach to implement into ModSAF the effects of suppressive fire. For subject variables, we assumed two levels of readiness. Level one is high in which the troops are well trained, have good leaders, and are fully equipped with the latest gear. Level two is low in which the troops are not so well trained, are plagued with poor leadership and low morale, and have older, poorly maintained equipment. To scope the effort, we chose three independent suppression variables to represent soldier interaction with suppressive fire: intensity of bombardment in terms of rounds per minute, proximity of detonation, and the location of the detonation (i.e., in front, to the side, or in back of the troops). The caliber of the round remained constant at 155 mm. The first step in the modeling process was to hypothesize a logical, defensible relationship among these variables. A multiplicative function was selected as a plausible first approximation. We know that as explosions get nearer, happen more frequently, and last longer (up to a half hour) the effects of suppression increase and the probability of selecting the best set of actions declines. Therefore, if a ModSAF DI is to emulate a soldier responding to suppressive fire, we believe the first effect to model is the decrement in the probability of selecting the best course of action (COA) for a particular set of conditions.

Assume for this discussion that the ModSAF entity experiencing suppressive fire is executing one and only one of a set of N COAs, such as an assault or withdrawal. In the baseline case, that is, without suppressive fire, the ModSAF entity periodically comes to a decision point and either continues its present COA or switches to one of the alternate COAs. At each of these decision points, the entity makes a rule-based choice that is always doctrinally correct. That is, at a decision point, the entity chooses the rule-based COA with probability 1 and chooses each of the alternate COAs with probability 0 when there is no suppressive fire.

The following approximate model of the effect of suppressive fire on the "appropriateness" of the choice of COA is based on the concept of a suppression index S that ranges from 0 to 1. The index S is calculated from the combined characteristics of the suppressive fire as perceived by the entity. The baseline value of S in the absence of suppressive fire is 0. Under this condition, the entity makes the correct rule-based choice at all times as above. When $S = 1$, the entity makes a random choice among COAs, thereby choosing the doctrinally correct COA with a probability of only $1/N$. All COAs are equally likely. This phenomenon may be represented mathematically as follows.

Let $P_{cor}(S)$ be the probability that the entity makes the doctrinally correct choice of COA. The following formula for $P_{cor}(S)$ is based on the assumption that P_{cor} depends linearly on S when S is between 0 and 1:

$$P_{cor}(S) = 1 - \left(\frac{N-1}{N} \right) S.$$

The overall probability of an incorrect choice is $P_{inc}(S) = 1 - P_{cor}(S)$. Let p be the individual probability of each incorrect COA. Then

$$p(S) = \frac{P_{inc}(S)}{N-1} = \frac{S}{N}.$$

To improve the model in the future, either data or the judgment of subject matter experts could be used to weight the alternate COAs when S is large.

The independent variables that determine the suppression index S are the intensity I of the suppressive fire in rounds per minute, the radial distance R from the entity to the impact point of the rounds in meters, and the azimuthal angle ϕ of the impact point relative to the entity's forward direction.

The suppression index S is assumed to be the product of individual functions of I , R , and ϕ :

$$S = j(I)g(R)h(\phi).$$

The functions j , g , and h each range from 0 to 1. With this choice, any one of the independent variables can reduce the effectiveness of fire suppression, as is reasonable. For example, if the distance to the point of impact of the suppressive round(s) is very large, then $g(R)$ will be 0 and the suppression index S will be 0 no matter how intense the bombardment. The three functions are defined in the following subsections for the case of 155 mm artillery fire.

Intensity of Fire

The influence of intensity of fire on the suppression index is given by the function

$$j(I) = \min[0.20\sqrt{I}, 1.0].$$

As I increases, j increases as the square root of I until it reaches its maximum value of 1.

Proximity of Fire

The effect of the proximity of fire on the suppression index is given by the function

$$g = \min\left[\frac{63}{(R+.01)}, 1.0\right].$$

The function g varies inversely with R until it maximizes at the limit 1.0 when R is less than 63 m. (The increment of .01 is added to R to prevent calculational difficulty if R is set to 0. It has little effect when R is greater than 63 m.)

Bearing of Fire

The bearing or direction of round impact relative to the forward direction of the unit affects the suppression index through the function

$$h(\phi) = 0.75 + 0.25 \cos\left(\frac{\phi}{2}\right).$$

With this choice of the function h , suppressive fire is most effective in front of the entity and least effective behind, with equal effectiveness on left and right. Note that direction alone cannot reduce the suppressive index to 0. The largest effect is a 25% reduction in the index. The coefficients in these equations will be different for weapons other than 155-mm artillery.

Due to the modular architecture of ModSAF, the capability exists to remove one module and replace it with a modified one. For this project, two modules or libraries received most of the attention, *libureactif* and *libuactcontact*. *Libureactif* deals with unit level reactions to indirect fire tasks, and *libuactcontact* focuses on unit level actions on contact tasks. Both deal with behaviors of entities when fired upon by weapon systems. Modification of the logic that a DI entity takes when fired upon by indirect fire is represented in these libraries. Further

development and integration of the methodology and equations produced in this effort will allow known deficiencies in ModSAF's representation of soldier performance to be corrected. Dealing with these shortcomings using this methodology will provide increased realism and appropriate uncertainty in DI performance prediction.

4.3 ModSAF Dismounted Infantry Team Operations on the DIS Battlefield

To understand how the suppression-sensitive synthetic infantry team would operate on the DIS battlefield, we produced the operations model in Figure 2 that illustrates how a DI might respond to suppressive fires given the mathematical argument above.

The first step in exercising the suppression sensitive ModSAF is to initialize various parameters of the simulation. These parameters are extracted from a scenario and include the mission to be accomplished, the enemy disposition, threat characteristics, the terrain, the weather conditions, and the critical mission times. As an example, consider the following scenario:

The action takes place on rolling wooded hills, on an overcast day with intermittent rain. The time is dusk. A Red company size force moves along the route of passage. Red scouts detect a Blue infantry platoon camped on high ground commanding the route of passage. Red opens fire on the unsuspecting Blue force. Blue first localizes the Red position 2000 m off and begins a hasty defense calling in artillery on Red position. Red comes under direct and indirect fire from Blue artillery and small arms. The engagement lasts for 10 to 15 minutes and ends when the Red force breaks off contact.

Variations on this scenario could be developed based on the duration, accuracy, and intensity of Blue indirect and direct fires.

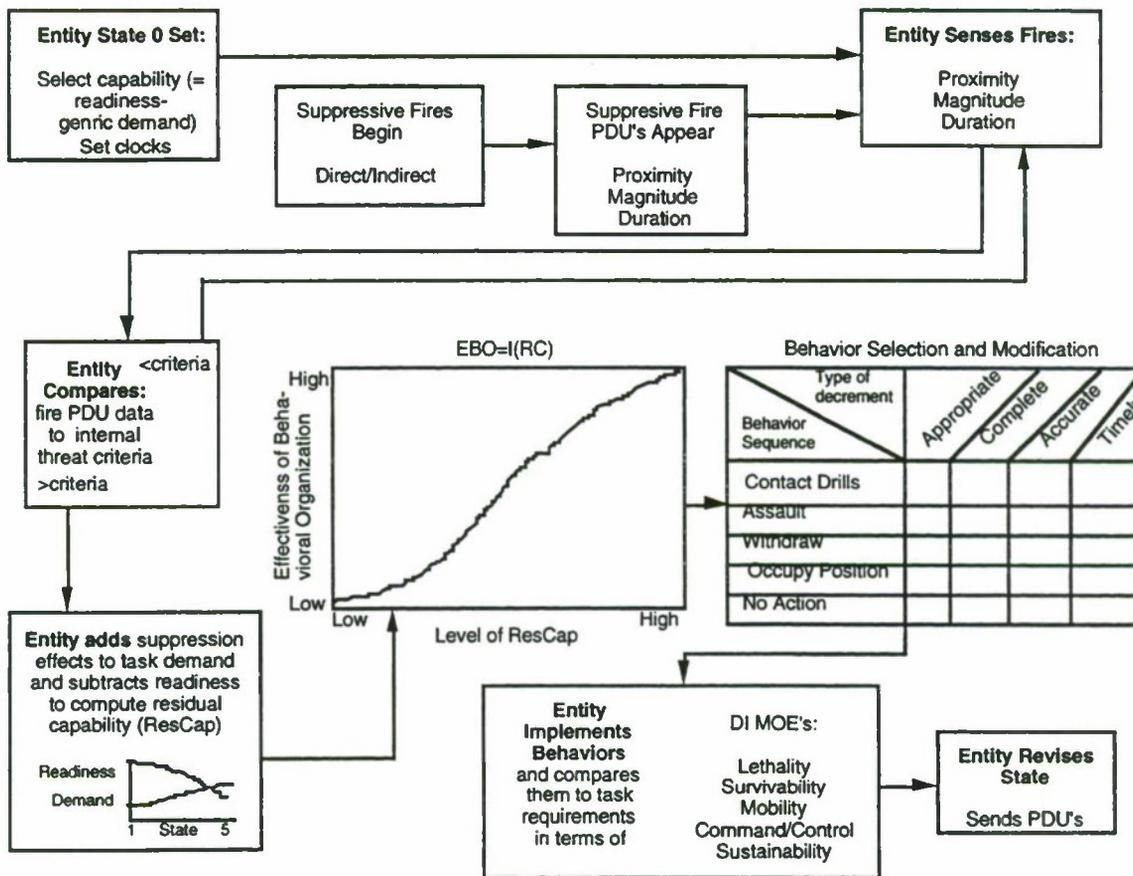


Figure 2: ModSAF DI Response to Suppressive Fires

The combination of parameters in the scenario establishes the initial generic "demand" on the ModSAF DI entity that corresponds to the demand in the UTB. The critical mission times are then input by setting the various clocks in the ModSAF entity. These times include mission start, beginning of suppressive fires, duration of fires, rate of fires, time of cessation of fire, and the time of residual suppressive effects (dependent on the rate and volume of fire).

The initial state of readiness of the ModSAF DI would also be set before the exercise begins. Readiness, as discussed earlier, is a product of individual and unit factors including level of training, role in combat, personal well being, cohesion, commitment, and leadership. Some of these factors are seen as "dials or gauges" on the ModSAF graphical user interface (GUI). Their values, or intensities, can be set in any number of possible combinations. Once the initial state of our suppression-sensitive ModSAF infantry team is set, the simulated suppressive fire can begin. Suppressive fires are initiated by a ModSAF artillery battery that sends out protocol data units (PDU) containing code that represents

the shell characteristics, initial coordinates, trajectory, and ending coordinates.

The latest development in ModSAF, made by the Leathernet support team at NRaD, represents suppressive fires by allowing virtual artillery to fire at predetermined points on the ground rather than only at targeted entities. These PDUs, which contain information on the point of impact relative to target position and the burst radius of the round are read by the sensitive ModSAF infantry team (SMIT). From these characteristics the SMIT calculates a figure of merit (FOM) that combines the intensity, duration, and proximity of the barrage and compares this FOM to a set of criteria that describe the minimal threat to the SMIT given its disposition, cover, and concealment. If the SMIT does not register the barrage as a threat, it goes back into its observation mode waiting for other stimuli. If it does register a threat, the SMIT calculates its own residual capability to continue its mission, given the level of the initial demand and the level of suppressive fire added to the initial demand. This calculation is made by comparing the combined demand on the SMIT to its initial

readiness. Demand is defined as the sum of all generic stress (environmental stress, mission workload, and specific stress from suppressive fires), and readiness is the total of all individual and social factors that make up a unit's capability.

The difference between readiness and demand defines residual capability (RC). If the demand is extremely high (mission impossible, bitter cold, high winds, mountainous terrain, and withering direct fire combined with intermittent artillery) it will exceed readiness no matter how well prepared troops are. This results in the development of "stress responses" in the SMIT whose analogues in real soldiers include such symptoms as inability to make decisions, stomach distress, panic reaction, etc. These "virtual symptoms" interfere with, disrupt, or impede the selection of behaviors that are appropriate to the task at hand. In the case of suppressive fires, behaving effectively, could mean taking protective cover until the barrage is over. The extremely high demand increases the probability that the behaviors chosen by the SMIT will not be appropriate to the task, may be carried out inaccurately if at all, and may take so long that they are ineffective. The outcome of this interference with the selection and implementation of combat behaviors will be to decrease mission performance in terms of lethality and survivability of the SMIT. On the other hand, if the demand is very low, there will be no noticeable decrement in performance with regard to implementing the proper course of action as dictated by the level of suppressive fire.

5. Conclusions

We developed a conceptual model of human behavior to explain relationships among behaviors in support of a taxonomy for synthetic entities. The taxonomy provides not only a classification scheme for abilities, tasks, and behavior descriptions but also a common language and syntax for representation of human behavior in synthetic forces within ADS. This taxonomy is compatible with DMSO's Conceptual Model of the Mission Space, STRICOM's combat instruction sets, the Integrated Unit Simulation System, and other ongoing behavior representation efforts. As a proof-of-concept, some of the behaviors of the phenomenon of suppression are being implemented in ModSAF DI. When completed, the suppression-sensitive ModSAF DI will provide the distributed interactive simulation

environment with the first synthetic entities that are responsive to the effects of weapons other than kinetic and blast injury. This accomplishment will open the door to representing other psychological phenomena such as fatigue and combat stress, in addition to the impact of radiological, chemical, and biological weapons.

6. Acknowledgments

This work is supported by the Defense Nuclear Agency and the Defense Modeling and Simulation Office.

7. References

- Blades, J.W. (1986). *Rules for Leadership*, National Defense University Press.
- Cannon, W.B. (1932). *The Wisdom of the Body*, Second Edition, New York, Norton.
- DeGreene, K.B. (1970). *System Psychology*, McGraw Hill, pp. 106-112.
- Directorate of Combat Developments (1979) *The Fort Sill Fire Suppression Symposium Report*, USA Field Artillery School, Fort Sill Oklahoma 73503.
- Dupuy, Col. T.N. (1987). *Understanding War: History and Theory of Combat*, Paragon House.
- Fineberg, Michael L. (1995). *A Comprehensive Taxonomy of Human Behaviors for Synthetic Forces*, Institute for Defense Analyses.
- Fleishman, E.A., and Quaintance, M.K. (1984). *Taxonomies of Human Performance*, Academic Press Inc.
- Hughes, Wayne P. Jr. (1995). *Two Effects of Firepower: Attrition and Suppression*, Department of Operations Research, Naval Postgraduate School, Monterey, Ca 93943.
- Levine J.M., and Teichner, W.H. (1971). *Development of a Taxonomy of Human Performance: An Information Theoretic Approach*, American Institutes for Research, Washington DC, AIR Technical Report TR-9.
- Meister, D. (1985). *Behavioral Analysis and Measurement Methods*, Wiley, New York.

Selye, H. (1956). *The Stress of Life*, McGraw Hill, New York.

----- (1955). *Stress and Disease*, Science, Vol. 122 pp. 625-631.

----- (1952). *The Story of the Adaptation Syndrom*, Acta, Inc., Montreal.

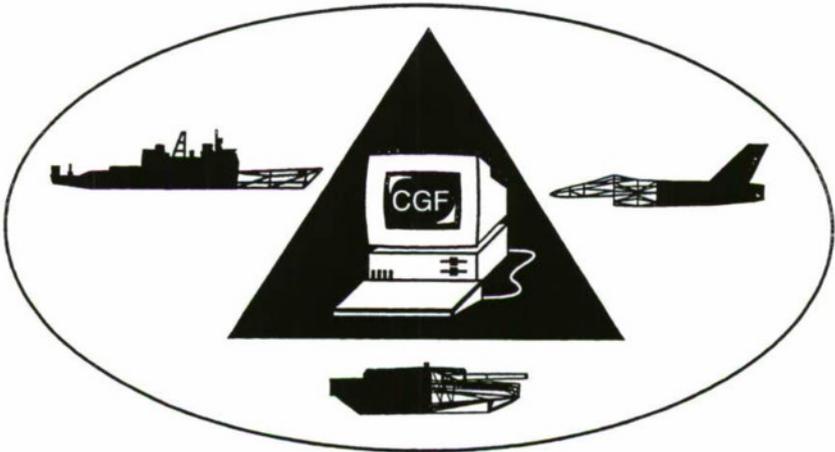
Dr. McClellan previously served as a staff physicist at Lawrence Livermore Laboratory and as a physics faculty member at the University of Maryland.

8. Biographies

Michael Fineberg is manager of Human Performance Technology Programs at Pacific-Sierra. He holds a Ph.D. in applied experiential psychology and a M.A. in Human Factors from the Catholic University of America. He has over 29 years of experience in the analysis, measurement, and enhancement of human behavior under stress. Dr. Fineberg can be reached at (703) 516-6251, fax (703) 524-2420, and e-mail at fineberg@sed.psrw.com.

Steven Peters, a Systems Engineer at Micro Analysis & Design, received his Bachelors of Science in Electrical and Computer Engineering from California State University at Northridge in 1981. He has over 15 years experience in software simulation involving Monte Carlo and discrete event simulations. He has simulated degraded soldier performance due to environmental stressors and integrated these human degraded performance algorithms into ModSAF, a Distributed Interactive Simulation (DIS) compliant combat simulation training and analysis tool. He has also developed and integrated a radar simulator into SIMNET-D, an earlier version of DIS. Steve can be reached at (303) 442-6947, fax (303) 442-8274, and e-mail at steve@madboulder.com.

Gene McClellan holds a Ph.D. in experimental physics from Cornell University. He is Manager of the Applied Physics Group at Pacific-Sierra Research, actively supporting the Nuclear Phenomenology Division of the Defense Nuclear Agency (DNA) from basic modeling techniques through the design of final products for the defense community. He directs software production for the assessment of human performance effects of ionizing radiation and biologic agent exposure. Dr. McClellan has directed a program for DNA to determine radiation doses to personnel involved in the Chernobyl nuclear accident and to learn from the Soviet experience of handling emergency operations in a high radiation environment.



Session 7b: Unit Control

Preston, Logica, UK
Landry, CAE Canada
McKenzie, SAIC
Penney, DRA, UK



Command Agent Technology in a War Game Simulation

Gary Preston
Technical Manager
Logica UK Ltd
Stephenson House
75 Hampstead Road
London NW1 2PL

Janusz Adamson
Project Manager
Centre for Defence Analysis
Fort Halstead
Sevenoaks
Kent

1. Abstract

The Centre for Defence Analysis, a division of the Defence Evaluation and Research Agency, has been researching the potential benefits of using Computer Generated Force (CGF) techniques within War Games and simulations, using its Divisional War Game (DWG) and the DRA's Generic knowledge-based Flexible Enemy (GEKNOFLEXE).

The DWG is a large scale, Divisional level, War Game running on VAX hardware. During a study, it is operated by approximately thirty military and civilian staff.

GEKNOFLEXE is an autonomous simulation running on Sun hardware which has achieved widespread recognition for its ability to credibly model Command and Control up to Divisional level using cooperating knowledge-bases.

The product of this research is the Command Agent Support for Unit Movement Facility (CASUM) which currently supports the DWG in all aspects of unit movement. Its principal function is to 'form up' a column of units from dispersed starting locations and then move them in single file to a specified destination while maintaining the order of march and unit spacing.

The aim of this paper is to provide an exposition of the following project issues:

- Aim & Objectives.
- Approach.
- Specification of Role.
- System Functionality.
- Benefits & Effectiveness.
- Potential Future Development.

2. Aim and Objectives

The overall aim of the Project may be concisely stated as:

"To investigate and report on the potential benefits of applying Computer Generated Force techniques within War Games and simulations."

This aim has been fulfilled by successfully completing a number of phased objectives. The Project has progressed from a simple demonstrator through to a CGF Facility supporting the DWG:

Phase 1 Simulation Interaction

Demonstrated a bi-directional interface between the two simulations including run-time consistency and reactive control of DWG units by GEKNOFLEXE.

Phase 2 Credible Control in Complex Task

Developed a Red Divisional Recce knowledge-base for GEKNOFLEXE and demonstrated credible control of Divisional Recce withdrawal within the DWG.

Phase 3 Capability in the Selected Role

Developed a unit movement knowledge-base and demonstrated credible movement of large groups of units within the DWG by GEKNOFLEXE under actual game conditions.

Phase 4 Release Version

Enhanced the unit movement knowledge-base to incorporate additional functionality and produced a release version of the CASUM Facility.

Phase 5 Game Support

Successfully provided game support for the 1996 game series.

The success of the Project was assisted greatly by building functionality incrementally, and by establishing confidence in the system through the series of demonstrators.

3. Approach

The technical approach was formulated to meet two basic requirements:

formulation of a simple development philosophy for the construction of a CGF Facility for the DWG. The relevant issues are presented in the following sub-sections.

3.2.1 GEKNOFLEXE

The original purpose of GEKNOFLEXE was to provide a generic system for the representation of a two-sided ground-based conflict at Divisional level and below. The intention was to develop a library of knowledge-bases which would enable efficient response to C3I study questions against the background of a conflict anywhere in the world and involving any two military forces.

Thus the emphasis of GEKNOFLEXE development was focused on the realistic representation and interaction of higher and intermediate level Command Cells. The level of unit aggregation, the degree of interaction modelling and the resolution of the terrain were all defined to serve this purpose and were fairly rudimentary.

For GEKNOFLEXE, the general approach to the development of a set of knowledge-bases would necessarily be top down. Representation would start with the highest level of command and be extended in depth and breadth until sufficient C3I modelling detail was incorporated into the system. Interactions between forces would be approximated algorithmically.

It is important to note that GEKNOFLEXE developments have always made full use of Command Agent Technology and, at lower levels where partial use of the technology would be valid, algorithmic approximations have tended to be used.

3.2.2 Command Agent Technology

Command Agent Technology is founded on the object oriented approach to the decomposition of a problem domain and supports an explicit representation of military command and control structures. It allows individual perceptions of the battlefield to be developed by each Command Cell modelled within a scenario. Systems employing Command Agent Technology can achieve high degrees of realism in the execution of the Command Cell function, and the interaction between Command Cells.

It must be stressed that the purpose of Command Agent Technology is to provide an appropriate framework within which to model C3I structures, and that, at the lower levels of command, the modelling of such structures may not require all the functionality provided by the technology. Hence a system may employ Command Agent Technology without fully exploiting its functionality. Further, within the paradigm of this technology, every entity on the battlefield is capable of being an 'agent of command' whether it is employed as one or not.

There is therefore, a distinction between the concept of Command Cells and Command Agents. Any entity modelled within a scenario can be a Command Agent - not all are Command Cells. The functional advantages provided by Command Agent Technology become more important as the level of command to be represented becomes higher. This does not however preclude lower levels of command (even sections) being developed using the same technology. In fact it is of advantage to do this since it facilitates extension and development of a system to include higher levels of command subsequently.

3.2.3 DWG

The DWG software is a single process simulation written in C which runs on VAX hardware. At the core of the DWG is the Simulator process which performs all the modelling and evaluation during the execution of a game. Game data are held in a DEC RDB database.

DWG players interact with the Game via individual player processes, separate from the Simulator process, which also run on VAX hardware. The player processes transmit orders to the DWG via Global Sections.

While the DWG and GEKNOFLEXE both address questions at the Divisional level, the DWG is not autonomous and supports a very low level of modelling detail. Further, this level of modelling detail is invariant and effectively defines the level of interaction with the system. In effect there is very little 'intelligence' built into the system and almost every task executed in the game needs to be micro-managed by the DWG players.

3.2.4 Development Approach

Given these issues it is clear that development of a CGF Facility must begin by applying control at the

lower levels of representation within the DWG. Successful implementation at this level has several potential advantages:

- Reduction of DWG manpower requirements through the removal of low level data-processing tasks.
- More efficient use of available manpower by allowing them to concentrate on higher level issues.
- Increasing confidence in the applicability of the approach to address higher level control functions.
- Subsequent, more sophisticated knowledge-bases will be able to utilise the work already done.

Consideration of these advantages led to a simple development approach for the CGF Facility, namely:

- Build the system and confidence incrementally.
- Address the simplest most time consuming tasks first.
- Provide 'Assistance' before 'Control'

4. Specification of Role

Promoting military enthusiasm for a CGF Facility to support the DWG was key to the success of the programme. It was therefore considered appropriate in the first trial to demonstrate the ability of CGF techniques to undertake a relatively complex task in addition to providing low level assistance. For the first demonstration of capability therefore, a Red Recce withdrawal was chosen and the military staff of the game were used to specify the knowledge base.

This decision was consistent with the general development approach because fairly low level reactive functions are undertaken during a recce withdrawal. Additionally, the need for recce to maintain maximum coverage of potential lines of enemy advance during withdrawal presented the opportunity to demonstrate higher level capability. The subsequent trial proved to be very successful and revealed a number of areas where a CGF Facility could provide valuable support to the DWG.

As a result, discussion with the military staff then focused on the most appropriate, initial application of the Facility. The rationale for the subsequent decision is developed in the following sub-sections.

4.1 Extent of Support

During the initial trial the military staff quickly recognised the potential of the system, but observed that most benefit would be obtained by addressing the more tedious and time consuming tasks first. This view was consistent with the development approach and it was agreed that the correct course of action was to initially limit the role of the Facility to that of an 'Assistant' to the DWG players.

In addition to meeting the development requirements this approach has some additional advantages:

- As an 'Assistant', the CGF Facility would have less responsibility and hence be less likely to adversely affect a game in the event of a serious failure.
- 'Assistant' level functionality would not require the representation of cognitive processes, hence making it easier to develop. Further, higher level 'Command' functionality would need to make use of 'Assistant' level functionality anyway.
- The DWG players would be free to address the more strategic and tactical aspects of their work.

4.2 Application Area

During the initial trial it became clear that the demonstrator was able to effect unit movement in a much more credible fashion than the DWG could by itself. Further investigation of the associated problems indicated that the most appropriate, initial role for the CGF Facility should be to comprehensively support the DWG in the execution of unit movement.

The background and rationale for this decision are presented in the following sub-sections.

4.2.1 DWG Movement

Within the DWG, a unit is moved from its current position to a specified destination using a DWG move order. This order allows the user to specify the required destination of the unit, whether or not to use available roads, a method for breaching minefields met, and a method for crossing rivers met, etc.

There is also a facility within the DWG for requesting a unit to follow a predefined route. A unit can be ordered to follow a particular route, but can only join or leave the route at a specified 'node'. The advantage of routes is that a number of them can be

identified and created before the game begins, removing the need for entering large numbers of move orders for units travelling along the routes.

However there are several inherent limitations within the DWG connected with unit movement, namely:

- **Naive Route Planning**

The DWG plots a straight line route between specified points. This takes no account of the intermediate terrain, so the route is not necessarily the fastest available, and it can pass through difficult terrain, possibly causing the unit to suffer casualties or halt. To get around this problem, the DWG player may nominate a number of waypoints, but this is a time-consuming and tedious process, especially in areas of inhospitable terrain.

- **Preparation of Predefined Routes**

Specific nodes along the routes must be identified prior to game start, and paths between these nodes must be planned. Routes are sometimes created during the game, incurring a further overhead in terms of data preparation time. There is also the chance that routes created during the game will accidentally pass through unfriendly terrain, leading to further time wastage.

- **Lack of Simple Movement Coordination**

If a DWG player wishes to make use of a predefined route, he still has to move all the units he requires in the convoy to appropriate entry nodes. These moves can encounter problems due to naive route planning. Once the convoy is in place, maintaining its momentum and Order of March is difficult due to units queuing.

- **Harsh Queuing Model**

If a unit attempts to move into a quadrant already occupied by another unit, then the moving unit may queue if the capacity of the quadrant is not sufficient to contain both units. If the obstructing unit does not move on, then the queuing unit will queue indefinitely. The DWG does not allow 'obstructing' units to be 'bulldozed' out of the way or simply bypassed.

- **Boomeranging**

This is a random effect in which a unit that has been 'magic moved' by an umpire may return to a previous location later in the game.

4.2.2 Rationale

Given the problems outlined above, and their effect on game realism and manpower requirements, it is clear why unit movement was chosen as the area to be addressed. A Facility to assist with unit movement would significantly reduce the workload of both players and umpires, and would lead to a greater emphasis on strategic and tactical thinking during the game.

5. Functionality

The primary function of CASUM is to provide a simple mechanism for DWG players to move large numbers of units around the battlefield in a coordinated manner. The overall functionality of the Facility however is much more comprehensive.

CASUM enables DWG players to move groups of units of any level in a coordinated fashion, with the minimum of effort. Movement orders given to a commander will automatically include all of his subordinates (not just immediate ones). Units that are not wanted on the move can be exempted from the move, and units that would not normally be amongst the commander's subordinates can be added to it. The single file convoy created from these units has a doctrinally correct Order of March (OOM) and units in the convoy are spaced appropriately depending on their size. A terrain-safe route planner that can accept a number of waypoints ensures that the movement of all units is not jeopardised by transit through difficult terrain.

Dispersal criteria can be specified that determine the circumstances under which the convoy should disperse when attacked. Units that disperse will head for cover and then camouflage. If an unexpected obstacle (e.g. a minefield or uncrossable river) is encountered, the convoy will halt and pass command back to the DWG player. If the convoy disperses or encounters an obstacle, it becomes suspended, and the DWG player can take any action necessary to resolve the problem (e.g. eliminate the enemy threat, or bring up an engineering unit to clear the minefield). The convoy move can then be resumed, and coordinated movement continues as before. DWG players also have the option to suspend the convoy voluntarily.

When a convoy has reached the end of its route, each of its constituent units deploys to cover in a formation that reflects the structure of the OOM.

When the whole convoy has deployed, command is passed back to the DWG player. The entire convoy move, or individual units on it, can be cancelled at any time.

The progression of the convoy from the initial forming up stage to the final deployment is conducted without the need for further involvement from the DWG player, unless an exception occurs. The amount of supervision required during the move is therefore minimal.

6. Benefits and Effectiveness

This section presents an evaluation of the effectiveness of CASUM in supporting the DWG task. Evaluation has included an assessment of DWG permanent staff comments, performance measures, and the overall effect on the DWG model.

6.1 Umpire Workload

Prior to the introduction of CASUM, the umpires' most time consuming task was solving queuing problems. With the introduction of CASUM umpires now have little to do bar monitoring of the game and providing advice to the DWG players.

It is clear from the games played so far that the role of the umpire has been significantly reduced by the use of CASUM. This reduction in role is sufficiently great to reduced the number of umpires.

Figure 2 shows the workload associated with the movement of a Brigade before and after the introduction of CASUM.

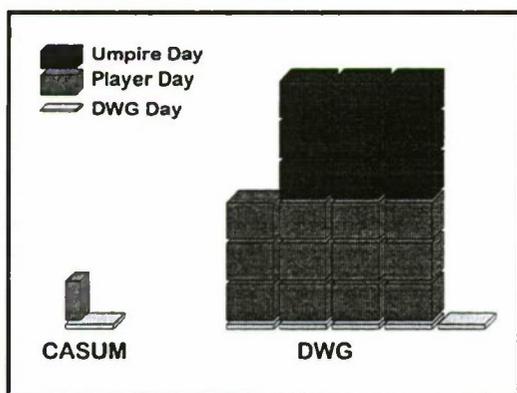


Figure 2 Effort Comparison

6.2 DWG Player Workload

The DWG players benefit from CASUM because they are now able to move large numbers of units easily without having to engage in extensive terrain analysis to determine suitable routes. As a result, they now have more time to deal with the tactical aspects of their roles rather than acting as traffic policemen.

The extra elements of realism introduced into the DWG by CASUM are of great benefit to the DWG players in that they replicate more closely the movement procedures on a real battlefield. The unit aggregations used in the DWG now resemble the unit aggregations used in real life. This helps to make the whole gaming environment appear less artificial, and enables the DWG players to apply familiar doctrines within the DWG.

Convoys can now be assumed to be moving correctly until such time as the convoy commander reports back to the DWG player reporting otherwise. A convoy coming under fire can now be relied upon to disperse to cover if the dispersal criteria are met. Consequently, units will not suffer excessively if an attack goes unnoticed. It is this extra 'intelligence' that helps alleviate the DWG players' workload.

6.3 Movement Realism

As well as removing much of the burden that the unassisted DWG previously imposed on the DWG players, the CASUM system has improved the realism of the movement model in the DWG. This improvement in realism is due to several aspects of CASUM functionality. These essentially provide default behaviour for units that in real life would do something sensible, but in the DWG do nothing.

6.3.1 Queuing Units

The solution of queuing problems was a major issue in the pre-CASUM DWG, and indeed constituted the primary part of the work of the umpires. CASUM deals with queuing in the DWG in a credible fashion - this is especially beneficial for units that are moving in convoy, which are particularly prone to queuing.

The existing queuing model is clearly at odds with reality, as a blocking unit would either move to the side of the road or be bulldozed out of the way. CASUM deals with this problem by forcing the unit into the next quadrant on receipt of a queuing

occurrence. This removes the requirement for umpire intervention.

6.3.2 Military Realism

CASUM also adds military realism to the DWG by allowing movement orders to be given to a commander only - his subordinates are automatically included in the move. This contrasts sharply with the previous state of the DWG where, say, a Brigade movement entailed the issuing of move orders to every unit in the Brigade individually.

The inclusion of different types of units (e.g. recce, armour, etc.) in convoy entities is more representative of the situation in reality, in that on the battlefield there are Brigade commanders, Battle-group commanders, and so on, not an artillery commander who deals with all artillery, or an engineer commander who deals with all engineers. Though the DWG is still based on this 'desk' approach, CASUM is forcing the various desks to cooperate in such a way that aggregated moves that would occur in reality are possible (and indeed encouraged) in the game.

The CASUM route planner prevents units moving into unfavourable terrain. This eliminates casualties due to going, which are extremely rare in real life but were common in the DWG previously because of the DWG route planner's penchant for moving units in straight lines, irrespective of the terrain.

6.4 Performance

Care is needed when trying to assess the impact of CASUM on the DWG. For example, when considering the efficiency of the system as a whole it is not sufficient to compare the amount of game time played with and without CASUM - if you moved one unit around the battlefield for an entire game, you would achieve a vast amount of game time but nothing in the way of useful results. In assessing the impact of CASUM more subtle measures of the effectiveness must be used. For instance, the total amount of ground covered during a game or the total number of events processed is a more appropriate measure of efficiency of the system. In the final analysis the important factor is whether CASUM adds military credibility to the development of a scenario and provides a more realistic environment for the DWG players.

6.4.1 Credibility

The introduction of CASUM has made the movement of all groups of units a much simpler exercise, and one which is more intuitive to the DWG players. This has freed up more of the DWG players' time for tactical thinking, with the result that the equipment being modelled in the DWG is being deployed and operated in a more realistic manner. This inevitably leads to more credible results being extracted from the system. Umpire intervention has been reduced drastically, which again improves the integrity of the system.

6.4.2 Movement Quality and Quantity

The quality of the movement conducted under CASUM is unarguably better than that conducted previously in the DWG. Units under CASUM control will never voluntarily go into terrain in which they may take casualties, and queuing will not bring numbers of units to a halt. Convoys will adopt a doctrinally correct OOM, and units will head for cover when under significant attack or when deploying.

The quantity of the movement conducted under CASUM is also unarguably greater than that conducted previously in the DWG. Comparisons with the last series reveal that approximately the same number of units have been moved but these units have covered twice the distance under CASUM.

7. Potential Future Development

The CASUM Facility is consistent with the approach to the development of an entry level CGF Facility for the DWG. Subsequent development must build incrementally on this. As stated previously, in its current form CASUM is considered to provide 'assistance' to the DWG player by undertaking his more mundane activities. For this reason CASUM is said to adopt the role of 'Commander Assistant', i.e. it assists the Commander in the execution of his task, but does not execute it for him.

It should be noted that to increase the level of CASUM functionality to that of a Command Cell, a great deal of lower level functionality must first be provided. It is therefore asserted that a wide range of 'Commander Assistants' should be available before the system is developed to support the more sophisticated aspects of command and control.

Consequently the recommended approach is to continue to develop 'Commander Assistants' for the DWG until enough have been implemented to warrant a shift to the development of Command Cells. At this point the Command Agent Technology will begin to be used to its fullest extent.

7.1 Commander Assistant Roles

Commander Assistant roles could address the following tasks:

- Artillery

The selection of deployment areas, allocation of guns to targets, specification of targets, and movement to new deployment locations.

- Engineering

Minefield laying/clearing, crater filling, and bridge building/blowing.

- Reconnaissance

In defence, coverage of all potential routes of the enemy advance until the main ones are established, withdrawal whilst monitoring these routes, suppression of the enemy at choke points by calls for artillery fire, RDM falls, and air strikes. In the advance, coordination of the movement of Reconnaissance forces to provide the optimal coverage of the enemy.

- Air Defence

Provision of an air defence screen around the front and flanks of a convoy and selection of suitable weapon states for the air defence units.

- Aviation

Plotting of routes that avoid enemy forces, radar searches at key points on a route, and identification and engagement of appropriate targets.

7.2 Unit Movement

Manoeuvre warfare is currently viewed as key to the achievement of military objectives, and hence development of the unit movement knowledge-base should be considered as an ongoing activity. Specific areas for development are:

- Route Congestion

Coordinate route usage to avoid congestion caused by a number of independent convoys reaching a river crossing point at the same time, or where terrain forces the canalisation of routes.

- Dynamic Route Planning

Re-assess the viability of routes as a result of bridge-blowing or cratering, constructing an alternative if necessary.

- Selection of Start-point

Select the start point for a convoy from a consideration of terrain, convoy length and position of constituent units.

- Response to Obstacles

Clear obstacles encountered by a convoy using convoy resources if possible.

- Deployment

Provide additional deployment pattern options to the DWG player at move specification.

- Orders of March

Provide additional OOM options to the DWG player at move specification.

7.3 Command Cells

Previous areas of suggested development are limited primarily to the role of 'Commander Assistant'. At some point however the sophistication of these 'Assistants' becomes sufficient that the C3I function of Command Cells themselves may be represented. Command Agents would be used to explicitly represent Command Cells, drawing on the Commander Assistant functions present within CASUM. At this level it is expected that the permanent Commanders would have the ability to specify sophisticated operations orders. The precise functionality that could be offered by such Command Cells would draw heavily on the experience gained from the Command Agent Research (CARE) project and GEKNOFLEXE work already undertaken by DERA.

8. Conclusions

In its current form the CASUM Facility is contributing considerably to the execution of the DWG task. There is significant potential for broadening the scope of the assistance provided and in the long term there is every reason to believe that the facility could be extended to fully automate divisional C3I functions. Further, the system has been received with much enthusiasm by the DWG military staff and is considered to have improved the credibility of gaming significantly.

In conclusion, the results achieved to date are considered to confirm that CGF techniques, particularly Command Agent Technology, have great potential to assist War Game simulations.

9. Authors' Biography

Gary Preston is the Technical Manager of the CASUM project. Mr Preston has a BSc(Hons) of Science degree in Mathematics. His previous projects include the Combined Arms Tactical Trainer Study and the Area Weapons Effects System Study for UK MoD. His research interests are in the areas of Simulation and Computer Generated Forces.

Janusz Adamson is a Senior Consultant at the Centre for Defence Analysis, DERA Fort Halstead. Mr Adamson has a BSc(Hons) degree in Astronomy and an MPhil. His project responsibilities include CASUM, the Close Action ENvironment wargame, Generic Algorithms, Real-time Knowledge Base Systems and Command Agents. His Technical focus is on Synthetic Environments and Computer Generated Forces.



Representative Communications for the Purpose of Command and Control in Computer Generated Forces

J. P. Landry, Eng., S. Valade, Eng., D. N. Siksik, Eng.
CAE Electronics Ltd.
St-Laurent, Quebec, Canada

1. Abstract

Communications is an essential factor of command and control. The effective representation of the communications process is therefore key to achieving a realistic command and control model for Computer Generated Forces (CGFs).

Experiment number five of the Aviation Warfighting Cell (AWC) program (U.S. Army Program Executive Office Aviation, Ft. Rucker, Alabama, February 1996) provided a vehicle to develop and integrate representative command and control communications into CAE's Interactive Tactical Environment Management System (ITEMS™).

2. Introduction

The involvement of CAE Electronics Ltd. in the U.S. Army Aviation Warfighting Cell (AWC) program provided a vehicle in which to integrate representative command and control communications into CAE's Interactive Tactical Environment Management System (ITEMS™).

This paper describes the design issues involved in generating this communications capability through ITEMS™ Computer Generated Forces (CGF) support of the AWC experiment.

A description of the AWC program is provided and is followed by a description of ITEMS™.

Emphasis is placed on the representation of communications networks within conceptual organizations such as teams, zones and commands. These organizations overlay one another in order to allow for: 1) individual players to talk to each other within a team; 2) teams of players to talk to other teams; as well as 3) commanders to talk to specific team leaders. This approach is aimed at providing a more representative model of the communications flow between command hierarchies on the CGF battlefield.

A discussion considering the characterization of

specific messages follows.

Basic messages such as spot and free-text reports provide an ability for CGF players to report threats detected by their sensors and to send, receive and act upon fixed message sequences.

The ability to implement command and control is supported by various status request messages such as position reports, fuel/ammunition inventories and "shot-at" reports. Other basic messages include move-to commands as well as target/control measure handoffs.

Control of CGFs is taken a step further with fire mission messages. These request/command based messages provide for coordination between players or between players and a manned simulator cockpit in order to establish laser designation and missile firing sequences.

Issues encountered which are particular to the simulation of communications in CGFs are described. They include the handling and correlation of threats reported by a manned cockpit, the acquisition of threats and the interaction, through free-text messages, of a pilot with his CGF counterparts.

Finally, applications of this technology are presented which include more robust training as well as an ITEMS™ CGF simulation with increased effectiveness in representing communications in command and control.

3. Aviation Warfighting Cell

The Aviation Warfighting Cell (AWC) (Holmes et. al. 1996, Larkin et. al. 1996) program is run by the U.S. Army Program Executive Office, Aviation. Its goal is to provide high fidelity distributed simulation of army aviation forces. The level of simulation will help to identify and study aviation issues into the next century.

To this end, the program focuses upon the creation of

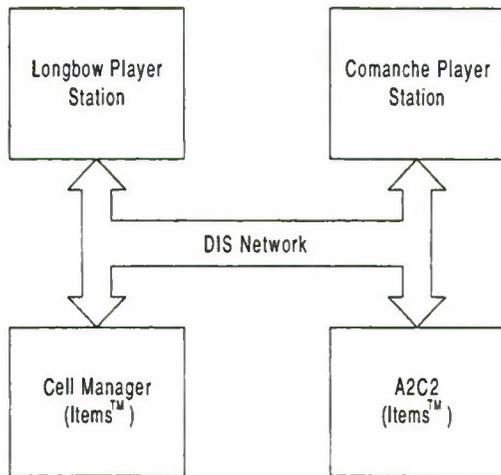


Figure 1: AWC Architecture

two full flight simulators: An Apache Longbow and a Comanche. The program studies interoperability issues affecting the communications between the two platforms. Experiment five of the Anti-Armor Advanced Technology Demonstration (A²ATD) experiments is referred to in this paper.

AWC consists of the two aforementioned simulators which are referred to as the Longbow Player Station (LPS) and the Comanche Player Station (CPS), respectively (see figure 1). ITEMS[™] provides Cell Management functions including overall control of the exercise and the provision of virtual forces to interact with the manned cockpits. Finally the Army Airborne Command and Control Vehicle (A2C2V) station introduces a man-in-the-loop command and control capability. All elements are networked via Distributed Interactive Simulation (DIS).

The simulation fidelity achieved on both the LPS and CPS is concentrated on several aspects of military aviation. These include flight dynamics, weapons procedures and performance as well as digital communications.

As the Cell Manager, the principal role of ITEMS[™] is to provide the AWC tactical environment. ITEMS[™] seamlessly integrates manned cockpits to its environment of computer generated forces. ITEMS[™] also provides session control allowing for data recording functions, after-action reviews and stealth view capability.

A command and control vehicle simulation is supported by the A2C2V. This ITEMS[™] based station acts as a battalion commander and is capable of communicating with the rest of the tactical

environment via digital messages. A2C2V visibility of the battle area is not global but is based upon information which it has gathered using its sensors and intelligence.

4. ITEMS[™] Overview

CAE's Interactive Tactical Environment Management System (ITEMS[™]) (Siksik et. al. 1994) provides simulations with an environment in which entities are closely modeled with respect to both systems and intelligence, allowing them to interact with each other and with their environment in a realistic manner.

The tactical environment provided by ITEMS[™] is both controlled and created by the user. Through the use of a database management system (DBMS), scenario files, each of which represent a complete tactical environment, may be individually created, modified or downloaded to the host-simulation computer to be run in real-time.

The scenario design function of ITEMS[™] is an off-line process involving the ITEMS[™] Database Management System (DBMS). During scenario design, the user provides the DBMS with the information required to simulate the tactical scenario. In order to hold large amounts of data efficiently, the DBMS is divided into individual libraries. Information about scenarios, players, systems, intelligence, etc. is stored in respective libraries as individual records. The libraries are organized in a hierarchical format so that high level libraries can reference the lower level ones. For example, the specifications defined for a Hellfire missile in the missile library may be referenced by an AH-64 helicopter within the Player Library. The helicopter may then be referenced by a scenario within the Scenario Library. Any scenario in this library may be executed.

Players represent the basic elements of a tactical scenario and are defined as any entity which has tactical importance. A player could represent elements such as tanks, trucks, installations, SAM sites, infantry, fixed wing, rotary wing, stealth craft, ships, submersibles, etc.

The ITEMS[™] scenario combines players within a terrain and visual database in which they may interact realistically with each other as well as with such environmental elements as weather or ground features.

In order to achieve realistic player interaction,

ITEMS™ implements representative modeling of:

- Dynamics
- Navigation
- Ballistics
- Systems
- Intelligence

This modeling is based upon detailed knowledge of physical player data. For its definition, a player references numerous low level library records within DBMS. These represent its platform and various systems. Platform definition is made with respect to physical characteristics, dynamic envelope and vulnerability. Systems libraries provide specification data for modeling and include: active and passive sensors (radar, FLIR, RWR, LWR, etc.), weapons (guns, rockets, missiles, bombs, etc.), countermeasures (flares, smoke, jammers, etc.), communications (voice, data, visual, etc.) and laser (designator, range finder). By combining platform and systems references, a large variety of computer generated players are created.

The creation of lifelike player behavior and reaction, however, requires the modeling of player intelligence and this, in turn, is based upon knowledge of player tactics (military doctrine). Tactics, whether used in air, ground or naval applications, require a specialized range of expertise. ITEMS™ exploits the ability of expert systems to carry out tasks of an expert nature thereby providing tactical capability to players.

Within the ITEMS™ expert systems, such tactical knowledge as mission, opponent selection, coordination and specialized combat is represented. This knowledge is represented by rules and provides control over the actions of individual players as well as over the summary actions of groups, such as a change of formation. Doctrines, like player data, are organized into libraries within the DBMS and are referenced by players within the scenario. This organization allows the user to assign different behaviors to players which share similar physical characteristics.

With the use of ITEMS™, a tactical environment in which players interact realistically is provided. It is desirable, via DIS, to network the tactical environment to both manned simulators and to other tactical environments.

5. Communications Networks

Simulation of tactical communications in a virtual environment is a complex problem. The model has to be physically correct by respecting the physical laws inherent to the communications medium (RF, optical, written, acoustic, etc.). Conversely, it must be tactically correct. Tactical organization of military communications is such that messages must only be received by the relevant military entities.

Directability of tactical messages is desirable for efficiency and for operational reasons. It reduces the communications traffic going through the onboard communications devices of the military vehicles and provides a basis to perform command and control as well as coordination.

By contrast, untargeted messages produce intense communications traffic which may result in mingled unintelligible messages, information overload or disruptive information analysis and the requirement of filtering by recipients.

Untargeted messages also introduce responsibility ambiguity which is antagonistic to military command and control. Recipients of a message cannot take direct responsibility for it, given that many other equally qualified entities may have received it as well. A targeted message, however, may be tailored to the abilities of the recipient. This facilitates the expedition of message interpretation by the CGF in the same way as it does in the real battlefield.

All ITEMS™ messages are transmitted and received through the simulation of the vehicle's RF onboard communications suite. Communications apparatus include analog/digital radios and direct links. These systems may be defined with encryption and eavesdropping capabilities. The models representing these devices take into account such factors as the emitter power, the antenna gain, the presence of jamming and the range between the transmitter and the receiver. This simulation is therefore physically representative and meets the first requirement of a tactical communications model.

The AWC program introduced an additional simulation layer to the ITEMS™ model by introducing the concept of tactical radio networks. Prior to AWC, ITEMS™ radio broadcasting was rendered directional via specific frequency selection on the radios of the entities that were meant to communicate together. Given physical radio connectivity, an entity could listen to the messages

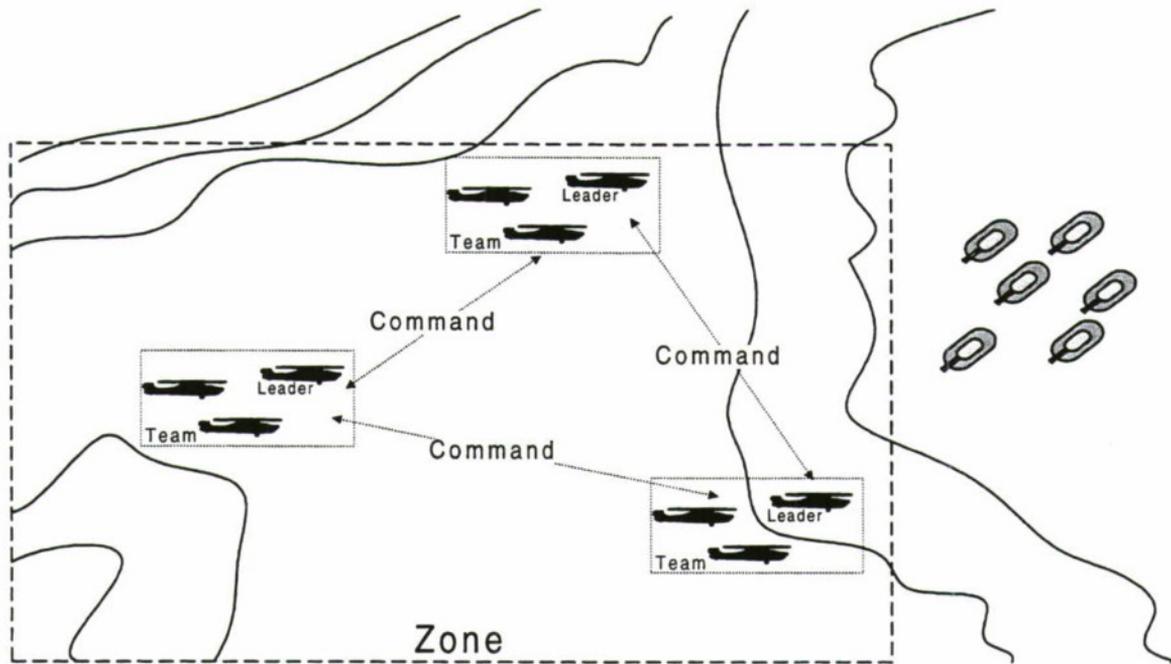


Figure 2: Communications Networks

transmitted on specific frequencies as assigned by its mission. The AWC experiment required ITEMSTM to expand on this functionality by providing for tactical networks. Messages were transmitted through Team, Command and Zone networks (see figure 2).

5.1 Team Network

The Team network is used to broadcast messages within a team eg. a vehicle unit, a formation or a platoon. The main application of this network is to facilitate intra-team cohesion and coordination. A team leader typically uses it for point to point communications with his subordinates in order to resolve the specifics of their mission. In turn, the subordinates use this network to report to their superior and to help him gather tactical intelligence.

5.2 Zone Network

The Zone network is used to transmit information to all teams within a contained geographical area. This network typically channels general broadcasts whose goal is to maintain uniform tactical awareness amongst the entire force in the zone.

5.3 Command Network

The Command network is used to network the team commanders so they can coordinate themselves for

specific inter-team tasks (eg: time on target fire, maneuver and support team coordination, observation and attack, etc.).

The integration of these networks within ITEMSTM makes it possible to simulate the real world communications methods that the A²ATD experiments required in order to study the interoperability between the AH-64 and the RAH-66. The networks map themselves to actual military chain of command. By doing so, they provide a communications infrastructure that allows information to flow to the proper entities or groups of entities.

From a CGF perspective, the simulation of networks enhances realism, directability and interpretability of messages. All of these qualities are required in order to have realistic CGF interoperability with manned cockpits. Moreover they help to represent more accurately the decision making process taking place on the battlefield. It is one thing to have virtual players take the appropriate action by performing good situational reasoning. It is another to achieve the same effect through the coordination of their subordinate counterparts.

The simulation of communications flow and of the decision centers is important to interoperability experiments such as A²ATD. It is likely to be more

important to future experiments that would analyze it at run time for intelligence gathering purposes. Command posts and observation posts can be recognized based on the volume of communications they generate and thus can be targeted first.

6. Communications Messages

AWC not only introduced the simulation of networks to ITEMSTM but also increased the number and variety of messages that ITEMSTM can handle.

6.1 Pre-AWC Messages

Prior to AWC, ITEMSTM radio messages were used to communicate battlefield information whose content facilitated uniform tactical awareness among the forces. From a content point of view, the messages are grouped into two categories: variable situational content messages and fixed tactical content messages.

Variable situational information content messages contain information about detected threats. These type of messages communicate such information as positions, velocities, level of acquisition, friend/foe status, etc.

Fixed tactical content messages typically report tactical events that other players can respond to. The nature of the event is not explicit in the message and must be interpreted in the behavior rules of the players receiving the message. These messages may be considered as coded since the receiver must have a prior knowledge of what to expect. Fixed tactical content messages do not include variable fields for the purpose of parametrization.

A sender of the types of messages described above has no control over how they will be interpreted by the receivers. A contact report details potential targets. It is not explicit about what should be done with them. A fixed tactical message is functional if both the sender and the receiver have the exact same understanding of what is implied.

The more explicit a message is on how to interpret the information it conveys, the better its chances of achieving efficient inter-player coordination. AWC experiment five was in many ways a coordination experiment. It had to demonstrate, given the communications bandwidth of the AWC digital messages, how well two helicopters with different missions and of separate manufacture (one of them a prototype) could coordinate between themselves and with their ITEMSTM generated virtual counterparts.

6.2 AWC Messages

Among the AWC messages used for experiment five are messages very similar to the ITEMSTM messages described above.

Spot reports are equivalent to the ITEMSTM contact report, with the exception that groups of vehicles (spots) are reported rather than a list of individual vehicles.

Free text messages are the equivalent of the ITEMSTM fixed tactical messages. As indicated by its name, free text message content is left to the sender to formulate as the specific need for communications arises. Its format is free and it therefore cannot be parametrized. It consists of a character string that a pilot can type in and transmit to other pilots or players. It is typically used to signal very specific mission events such as reaching a position or flagging the beginning of an action.

Usage of the free text messages, implies that both pilots and virtual players must have briefing knowledge about which free text messages to use under specific circumstances. As with the ITEMSTM tactical messages, interpretation of free text messages by virtual players is described by their behavior rules.

Message introduced to ITEMSTM by AWC convey information required for the command and control of virtual players from the manned cockpits.

Information query messages are used to obtain information from virtual players about their own state. Players respond automatically to these messages. Query messages include status reports (fuel level, ammunition), present position reports and shot-at reports (list of weapons fired to date along with the targets).

Since query messages are explicit about the action to perform, an automatic response can be formulated.

Newly developed messages included requests for which automatic behaviors were modeled. An example is the move command which requests a route change from ITEMSTM players.

An additional message capability integrated to ITEMSTM provides for the interpretation of control measures. The fire / no fire zones messages are examples. Based upon their opponent selection rules, the receptivity of virtual players towards selecting opponents inside or outside of specific areas may be

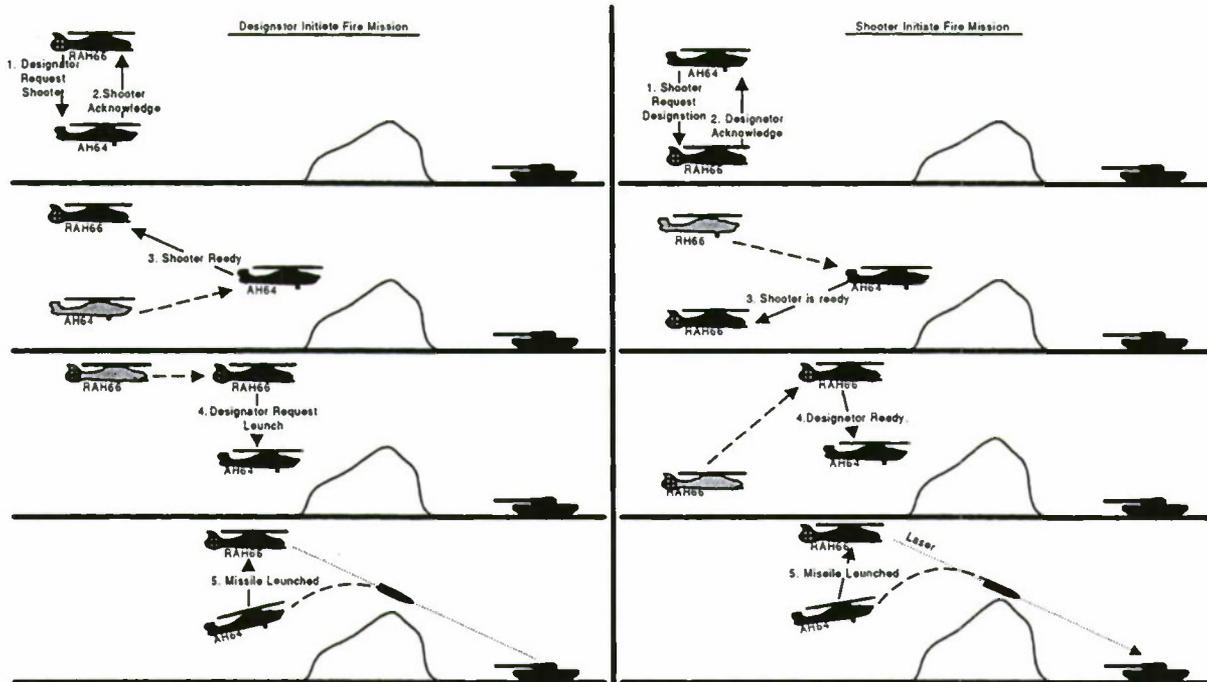


Figure 3: Coordination of Laser Designation and Missile Launch for Fire Mission Message

varied.

A level of complexity above those already mentioned, is a family of partially automatic interpretation messages. Partially automatic messages imply that the receiver is subjected to a bias towards acknowledging the request. This bias is introduced via the opponent selection process. In this way, a designated target, provided via communications, may be selected or rejected, based upon the bias.

ITEMS™ players can therefore base their receptivity to these messages upon tactical context. Once selected, however, the actions to perform with respect to the target are fully automatic. These actions do not need to be described within the player's rules since they are explicitly detailed in the message. The target handover and fire mission messages are messages in this category.

The target handover message describes which target to engage and which weapon to fire. Automated responses include the maneuvering (approaches, unmask, etc.) required to attain a firing position as well as the request for fire itself.

The fire mission message is a multi-step communications protocol which permits the coordination of laser designation and missile launch between two helicopters. This protocol addresses the time delay problems, the maneuvering, the laser code transmission and target reporting in order to laser the

target in the last seconds of missile flight only (see figure 3). Provision was made for this message to be used as an artillery request when required. It should be noted that because of a lack of time, the fire mission and artillery request messages were only integrated within ITEMS™ and not to the AWC cockpits.

In order to consider communications failures and the uncertainty they may cause, an automatic acknowledge message has been implemented. The user may select a mode that will enable all players to expect an acknowledge when they send a message. A player will re-issue a message up to three times to players that did not acknowledge reception of his message.

7. Integration Issues

The integration of manned simulators with any CGF always raises correlation issues. These are typically related to such considerations as the difference in fidelity for sensors and weapons, inconsistencies between different scoring models, the lack of correlation between separate terrain databases, etc.

During the AWC experiment, the use of U.S. Army approved algorithms addressed most typical correlation issues. This section will focus on the correlation issues related to communications and their impact on the simulation of command and control.

The correlation issues discussed pertain to message data, effects and interpretation, the use of acknowledge messages in order to achieve better situational correlation for pilots and the necessity of using free text messages to address mission details which are too specific to parametrize.

In order to achieve interoperability with manned cockpits, ITEMSTM was required to be able to correlate the data received by players via messages with data that players had collected individually, via their sensors. As an example of a possible lack of correlation, a spot report might report one too many vehicles or may contain errors in the attributes of the reported threats.

In order to solve this problem, ITEMSTM players associate reported entities with their closest currently detected threats via a weighting mechanism that would increase in certainty based upon the amount of data that could be matched (ie: positions, speeds, player types, level of acquisition, etc.). Entities that cannot not be matched with already detected threats will then be matched to possible yet undetected threats. If ITEMSTM determined that no matches were possible (eg: an impossible position is reported), the information is discarded. In all cases, the receiving player preserves the most recent information about the reported threat.

An example considers a player receiving information about a threat it does not yet detect with its own sensors. This player will use the message data in order to build its own internal image of the threat. As soon as the player detects the threat with its onboard sensors it refreshes this image with its own data. For specific data, if the message information proves to be more accurate, it will be retained.

Another data correlation issue arises as data contained in messages starts to age. Players reported some time ago may have been killed or simply changed location. A solution to this kind of problem is to extend the life of the data by extrapolating it over time. ITEMSTM extrapolates the location of the threats based upon the speed at which they were last reported or detected. This solution is valid for a limited time, however, and for this reason, ITEMSTM also tags the information with a validity timer (specific to the player type) beyond which the reported data is discarded. Data describing a slow moving threat such as a tank will be extrapolated for a longer period of time than data representing a fighter aircraft since magnitude of the potential extrapolation errors increases dramatically over time

for the latter.

The network discussion in section 5 highlighted the effect of the directability of messages on their interpretation. For the sake of both command and control simulation as well as interoperability with manned simulators, the correlation of the meaning of the messages themselves is also very important.

A distinction must be drawn between the effect that a message will have on the battle and the reason for which it was sent. The variability of the effect of a message depends upon the design of a player's mission.

A player having a very precise mission can operate on its own and although it could accept command and control messages it may not require them in order to achieve its goals. A simple threat report may be the only triggering event required for its mission actions to begin.

Conversely a player with no precise mission will essentially do as it is instructed. This situation is far more demanding from the command and control point of view as it implies that explicit requests are required from command.

Integration of players with both high and low levels of mission detail is a delicate task which depends upon the command and control abilities of the scenario builders as well as their coordination with the pilots of the manned cockpits. The pilots require a clear understanding of how to inter-operate with the virtual players during missions just as they must know how to inter-operate with real forces in real life.

One focus of an ITEMSTM mission description is the opponent selection and mission rules associated to players. These rules automate the reactions of players based on events. Consequently, for the case of messages that are fully explicit about the nature and sequence of actions to undertake, it became evident that correlation errors could be avoided by removing their interpretation from the rules. The concept of macro rules was introduced in order to fully automate message responses, thus relieving the rule creator from having to repeatedly enter rules to handle the event of message reception. In order to deal with cases for which the correlation of the message and the action required is not as straightforward and must be resolved differently depending on the tactical context, the interpretation was left in the rules.

There exists a need for virtual players to correlate the

content of the messages they receive with their vision of the battlefield and with the way they intend to achieve their mission. This needs exists equally for pilots of manned cockpits. The pilots' sole link to their virtual partners is the family of digital messages they can exchange. During the integration of the cockpits to the ITEMS™, it was apparent that the pilots required confirmation that their messages had been received by their virtual partners. For this reason, the acknowledge message was implemented. This message gives tactical confidence to the pilots that everything was under control on the virtual side. The pilots could even assess casualties in their team when a subordinate player would fail to acknowledge the reception of its messages.

As mentioned earlier, the use of the AWC free text messages implies that both the pilots and the virtual players must be briefed about which free text messages to use under specific circumstances prior the exercise. The potential for correlation problems is therefore very large. A simple typing error from a pilot would result in virtual players failing to decode the message. For this reason, it was decided at the early stage of AWC integration that ITEMS™ would not support the reception nor the expedition of free text messages both to and from the manned cockpit. It soon became evident, however, that in order to handle very specific scenario details, the use of these messages was indispensable. Virtual players transmitted free text messages to signal their leader (a manned cockpit) that they had reached their fire position. They then received free text messages from the pilot as final confirmation before firing.

8. Conclusions

Through the use of ITEMS™, the AWC experiments have provided for significant improvement in the ability of man-in-the-loop simulators to interact with computer generated forces through command and control communications.

The implementation of networks, such as team, zone and command, allows for better directability of battlefield data.

Automatic specific action messages, such as "move-to" as well as coordinated messages such as "target handover" allow for improved control of subordinate forces during a mission.

Free-text messages provide the flexibility to increase the scope of interaction between manned cockpits and the CGF.

Although issues involving correlation of message data and interpretation remain, the work achieved for AWC experiment five provides a strong foundation for an increased representation of command and control in ITEMS™ as well as for an improved training capability for man-in-the-loop simulators.

9. Acknowledgment

The authors would like to acknowledge the dedication and hard work of the entire ITEMS™/AWC team, without whom this paper could not have been written.

10. References

Holmes, R., Representative CGF Behavior in a DIS Environment Through Digital Communications, SimTecT 96, Melbourne, Australia, March 1996.

Larkin, M., Ferranti, M., Obear, P., U.S. Army Aviation Warfighting Cell Anti-Armor Advanced Technology Demonstration Experiment 5.

Siksik, D., Beauchesne, G., Holmes, R., The Integration of Distributed Interactive Simulation Protocol Within an Interactive Tactical Environment, Royal Aeronautical Society, London, 1994.

11. Authors' Biographies

Jean-Philippe Landry is a Systems Engineer at CAE Electronics Ltd. and is primarily involved in CGF behavior simulation. Mr. Landry holds a Bachelor's degree in Engineering from McGill University.

Stéphane Valade is a Systems Engineer at CAE Electronics Ltd. and is involved in tactical equipment simulation. Mr. Valade holds a Bachelor's degree in Engineering from Ecole Polytechnique de Montréal.

David N. Siksik is a Group Leader at CAE Electronics Ltd. and is responsible for ITEMS™ behavior simulation. Mr. Siksik holds a Master's degree in Engineering from Concordia University.

An Architecture for Integrating Command and Control Capabilities of Heterogeneous Simulations

Frederic McKenzie
Science Applications International Corporation
3045 Technology Parkway
Orlando, FL 32826
Rick_McKenzie@cpqm.saic.com

Gregory Shumaker
Science Applications International Corporation
shumaker@nefarious.orl.saic.com

Pete Campbell
Science Applications International Corporation
campbelp@vorlon.orl.saic.com

1. Abstract

The Advanced Distributed Simulation Research Team (ADS RT) at SAIC-Orlando has been conducting experiments with the interoperability of simulations. Two of these experiments and their common architecture are discussed in this paper. The first of the experiments involves a linkage between Command Forces/Command & Control Simulation Interface Language (CFOR/CCSIL) and Combined Arms Tactical Trainer - Semi-Automated Forces (CATT-SAF).

Close Combat Tactical Trainer (CCTT) SAF and CFOR Command Entities communicate using CCSIL messages making CATT-SAF a player in CFOR activities and allowing a wide variety of doctrinal CATT-SAF behaviors to be available to CFOR. The focus here was to employ commanders using CCSIL to command and control CATT SAF entities. The proof-of-concept implemented consisted of a translation of the CCSIL Execute Directive message and the actions it could elicit as of version 3.1.1. A full mapping of CCSIL messages to CATT-SAF behaviors and a capability of sending reports from CATT-SAF to CFOR is the next step in this research

The second experiment explored interoperability between SAF simulations and live Command, Control, Communications, Computers and Intelligence (C4I) systems, such as Phoenix. The proof-of-concept implemented shows CATT-SAF units commanded from Phoenix via US Message

Text Format (USMTF) messages. This capability allows a common Command and control (C2) interface employed in the Army's Battle Labs (Phoenix) to be utilized as a front-end to CATT-SAF. The version of Phoenix used did not have the command order capability that is present in Phoenix version 1.1 which will need to be examined before a proper mapping of Phoenix messages to CATT-SAF behaviors can be completed. Also a capability of transmitting reports from CATT-SAF to Phoenix should also be pursued.

The architecture used CORBA technology to provide common services for command language communications among the simulations. This paper provides an explanation of the experiments and a description of the architecture involved.

2. Introduction and Background

The ADS RT at SAIC-Orlando has been conducting experiments involving the interoperability of simulations. For each of these experiments in interoperability, an approach was taken based upon a common architecture. The experiments discussed in this paper involved the integration of additional C2 capabilities into dissimilar simulations. The integration of these capabilities involved the development of a common architecture through which the command and control information is sent, received, and translated into the format used by a given simulation.

The common architecture for each of these experiments is based upon the Seamless Interoperable Simulation Environment (SISE). SISE was developed for our interoperability experiments in order to provide the object-oriented plug-&-play environment necessary for the interoperability and reuse of legacy simulation components. The use of SISE also provides insight into interoperability issues leading to a roadmap for the interoperability of future simulations. Each interoperability experiment uses some subset of SISE. Although portions of SISE have been implemented, it is currently a paper concept.

Interoperability in simulation can be achieved by adhering to common industry standards. Standards which encourage or require object oriented design are important not only for purposes of interoperability but also for the purposes of reusability. Distributed Object Technologies (DOTs) typically provide such a standard through a common framework for distributed objects. The interoperability experiments described in this paper make use of DOT technology as well as other standards.

The goal of the first interoperability experiment involving C2 was to provide a mechanism by which CCTT-SAF can receive and interpret CCSIL messages. This allows for the control of CATT-SAF entities by a CFOR commander or any commander with the capability of sending CCSIL messages.

The second interoperability experiment involving command and control provides the capability for CATT-SAF to receive command and control information which was sent in USMTF by a commander at a Phoenix console. This capability demonstrates not only interoperability but also the reuse of this common C2 interface.

2.1 SISE

SISE (pronounced size) offers two distinct capabilities. First, it offers a library of legacy simulation component services that have been wrapped for reuse. SISE allows these simulation pieces to be connected together to form a whole simulation or to be linked into an existing simulation. Second, SISE offers a mechanism for allowing simulations of different types (live, virtual, constructive) to interoperate. To support these two capabilities, different standards and different representations will need to be arbitrated. Hence, SISE contains the following arbitrators: semantic,

protocol, temporal, and spatial. These arbitrators will be elaborated in subsequent publications.

SISE is composed of three distinct parts: the core, the plug, and the GUI. Together, these parts allows a set of simulations and/or simulation components to interoperate. For the research presented in this paper, SISE provided a testbed for interfaces with CCSIL Signal PDUs and CORBA technology.

The SISE core consists of the SISE infrastructure and an object model. It is anticipated that SISE will one day be conversant with the High Level Architecture Run-Time Infrastructure (HLA RTI) thus providing a means by which legacy simulations can participate in HLA exercises.

A SISE plug is a client interface to a simulation, or a simulation component. The plug allows a simulation component to talk to the SISE core and to other simulation components participating in the exercise (participants). The plug contains an interface to the core which includes registration services, services for synchronization with the simulation clock, and queues for sending and receiving messages from other participants.

The SISE GUI provides the user with an interface to the library of services that have registered in or are defaulted with SISE. The GUI receives descriptions of simulation components via the core. The user may then use the Simulation Integrator portion of the GUI to construct a simulation. Once the simulation is constructed the user may activate components and start the simulation. Through the GUI, the user can create entities, examine the status of entities, monitor events, and control the simulation clock.

Through the SISE GUI, any combination of point-to-point simulation component connections can be emulated without having to write specific conversion routines every time for every combination. Also, when new simulations are added with SISE plugs, little work needs to be done to previously connected simulations to allow them to participate.

Portions of SISE that have been wholly or partially developed and tested to date are the GUI and parts of the infrastructure that allow plug-&-play, semantic arbitration, and protocol arbitration.

2.2 Distributed Object Technology

Unlike traditional distributed simulations that use distributed processes exchanging messages, object-oriented simulations are concerned with distributed objects and use them to provide the communication between heterogeneous distributed environments via remote method invocations. Interoperability is possible because the objects conform to the representation and interfaces needed by the host application, regardless of whether the object is local or remote.

Object Request Brokers (ORBs) provide facilities that allow applications to invoke object methods and receive responses from objects created and owned by remote servers. All ORBs define an interface such as the CORBA Interface Definition Language (IDL) that allows remote objects to be seen in an identical fashion to the application as a local object, without regard to the object's true location.

As mentioned previously, all ORBs must define an interface for interoperability. The Common Object Request Broker Architecture is designed to do just that (OMG 1995). It separates the interface of an object from its implementation using an IDL. The ORB will generate proxy objects for every remote instantiation of an object needed from a server. The ORB processes an object method invocation by first locating the object, invoking it (starting a server), converting the data as necessary, invoking the corresponding object method, and processing the return data. For example, an application may request pointers to remote objects and then use these pointers to call the remote services of the objects. The ORB provides a reliable communication path between heterogeneous applications using the same objects.

The use of object-oriented technology in the DIS and CGF simulation domains is a relatively new concept. However, some previous work has been done in the area. SimCore uses CORBA objects to encapsulate PDUs for internal communication using ORBs and ODBMSs (Lander 1995). Kuhl also has encapsulated PDUs and used these objects via an ORB for air traffic control (Kuhl 1994). The ORB was used to provide the service needed by the Entity Update Service of the DIS protocol. The Entity Update service of the ORB provides the publish and subscribe service needed by the SISE GUI mentioned previously. Object-oriented technology is being used in a different aspect by Sureshchandran to manage dynamic changes in the environment (terrain, weather, etc) to provide real-time, high fidelity environments on demand (Sureshchandran 1995).

The Tri-service Advanced Countermeasures and Threats Integrated Combat Simulation (TACTICS) also uses an object-oriented environment to allow high fidelity simulations to interact across distributed platforms. (Peck 1995). Similar to the SISE plug concept, TACTICS provides for the migration of existing simulations to the object-oriented model but focus on simulations that assess ground combat vehicle survivability.

The Joint Precision Strike Demonstration program (JPDS) is using advanced real-time ODBMS technology to facilitate the logging and analysis of DIS PDUs, tactical messages, and audio/visual data. The logged information can be accessed during run-time or during an After Action Review.

2.3 CATT-SAF

Close Combat Tactical Trainer (CCTT) is a distributed training system that simulates battlefield aspects of the US Army SAF, man-in-the-loop simulators, and indirect fire and logistics components. The SAF component of the CCTT, referred to here as Combined Arms Tactical Training SAF (CATT SAF), simulates US Army and Soviet Army tactical behaviors for dismounted infantry, vehicle, platoon, company and battalion echelons. These behaviors are developed from documented and validated Combat Instruction Sets (CISs) and, as such, are useful for training, mission rehearsal, acquisition, and Test and Evaluation (T&E) environments.

As part of the object-oriented design, CATT SAF uses a SAF Entity Object Database (SEOD) which contains representations for units, task organizations, overlay symbols, execution matrices, commanders' orders, and subordinates' reports. It is through the SEOD that this research influences the CATT-SAF simulation.

2.4 Phoenix

The Phoenix Battle Command and Control Decision Support System (BCDSS) is increasingly being used in the U.S. Army Battle Labs. This system has been successfully used to provide a user friendly human interface for the voluminous amount of information received by high echelon military personnel. This live, go-to-war system aids the decision maker in managing data sources so that more rapid command of forces may be effected.

Phoenix is intended to link with a number of systems such as IVIS. The primary interface employed in this research is an email capability that uses USMTF messages. These messages are used to convey reports to the commander as well as track enemy and friendly forces. The version of Phoenix used in this research did not have an Operational Order (OPORD) capability used to generate and transmit orders to subordinate units. Therefore, USMTF freetext messages were used for this purpose.

2.5 CFOR

Command Forces (CFOR) is a program under the Synthetic Theater of War (STOW) effort sponsored by the Defense Advanced Research Projects Agency (DARPA) that is intended to emulate high echelon commander reasoning and decision making using a variety of information sources including sensory capabilities in the synthetic battlefield. These command entity reasoners communicate with simulations as well as other reasoners by using the Command and Control Simulation Interface Language (CCSIL). CCSIL provides reports and orders information for all aspects of military operations including ground, air, and sea. These messages are communicated across a Distributed Interactive Simulation (DIS) network encoded within Signal Protocol Data Units (PDU). A number of CCSIL messages have been defined and an interface to ModSAF have been provided by The Mitre Corporation that allows an automated commander to control units in ModSAF.

The work performed in this research adapted CCTT SAF to accept CCSIL messages and therefore provides an additional simulation capability for CFOR. This proof of concept used the CCSIL Execute Directive message to convey commands to a CCTT unit using CORBA technology.

3. Common Architecture

Both of the experiments make use of a common architecture based conceptually upon SISE. A plug was built for each of the simulation components in the experiments. The plugs in this architecture do not contain all of the features required of a SISE plug, e.g. the registration mechanism was not included for each plug. Although this is the case, the SISE plug concept is still intact. Each plug is in

essence a client interface between a simulation and SISE. The plug sends simulation information to SISE or SISE may grab this information from the DIS network as in the case with CCSIL Signal PDUs. SISE then communicates this information to various services via an ORB. These services are translators to a generic command and control format. Upon reaching their destination component(s), the messages in the generic format are converted into the components' native formats. In the case of CCTT SAF, the native format would be SEOD format.

This architecture allows any compliant simulation to send and/or receive command and control information from any other compliant simulation. Compliance only requires the construction of an interface plug and a translation service (if a translation service does not already exist). When linking many components, this approach is much simpler than providing direct linkages between each component, requiring only $O(n)$ translations versus $O(n^2)$ translations.

3.1 Generic Format

The generic command and control information format used is based upon a small subset of command and control information. The command and control information sent in these experiments is a subset of the CCSIL execute directive message. This message contains orders for change formation, execute action drill, contact drill, halt, resume, and others. This set of orders was chosen because it consisted of a small set of messages that would be simple to implement while yielding practical, visible results.

4. Phoenix to CATT-SAF Implementation

The first experiment involved linking both Phoenix and CATT-SAF to allow a commander at a Phoenix console to send orders to CATT-SAF units without learning the CCTT SAF Workstation GUI. The intent of this experiment is to demonstrate the reusability of the Phoenix console and the construction of a "proof of concept" of using an ORB to provide the interoperability between simulation components in a manner correlating to SISE. The ORB used was a COTS implementation called Orbix by Iona Technologies. A diagram of the implementation can be seen in Figure I.

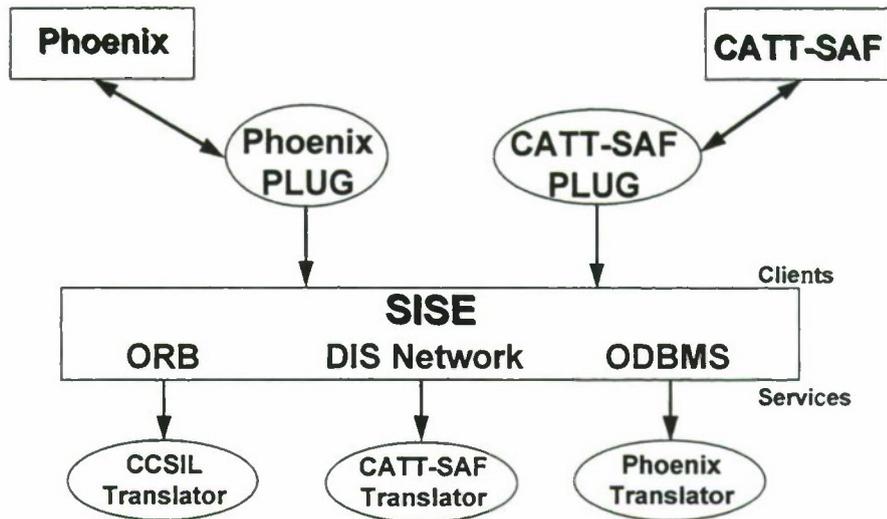


Figure 1: Phoenix Linkage

The Phoenix mechanism used for sending command and control information in this experiment was a USMTF message sent via e-mail. When the e-mail message is received, a process is executed which uses the message as input. This process is the SISE plug for Phoenix. It is implemented as an ORB client and calls a translator service which translates the USMTF into the generic command and control format. After performing the translation into the generic format, the information is placed into an order object. The order object is stored in the ORB until it has been delivered to its destination(s).

On the other end is a SISE plug for CATT-SAF. This plug is responsible for the reception of all command and control messages intended for CATT-SAF entities. This process theoretically should receive a callback when the generic order is placed in an Object-Oriented Data Base Management System (OODBMS). Since this portion of the architecture was not implemented, the process ticks the ORB periodically to see if there is an order for its entities stored in a generic order object. If there is, then the order is translated into the equivalent CATT-SAF order and placed into the SEOD appropriately for use by the current execution of CATT-SAF.

At the time of implementation, there was no Ada ORB implementation available. Thus, for purposes of translation, the CATT-SAF plug was implemented

in Ada with the ORB client portion implemented in C++. Thus, an Ada main program was executed with an interface to the C++ ORB client routines. This prevents the C++ runtime libraries from being initialized upon execution of the process. This is also why the plug was implemented to tick the ORB periodically rather than having a callback function associated within the ORB to appropriate messages to the CATT-SAF plug upon reception.

5. CCSIL to CATT-SAF Implementation

The second experiment involved the linkage of CCSIL messages to CATT-SAF, thus allowing CATT-SAF to play in CFOR exercises. Once again, the only CCSIL message that was used in these experiments was the execute directive message. This implementation is illustrated in Figure 2. Note that almost all components from the previous experiment were reused.

A CCSIL capability was implemented within SISE which pulls Signal PDUs off of the network and strips out the CCSIL messages. The CCSIL messages are sent out through an interface that is part of the SISE-GUI. The plug then calls the translator service to translate the messages into the generic format discussed previously. The generic message is then forwarded to the CATT-SAF plug in the same manner as the first experiment.

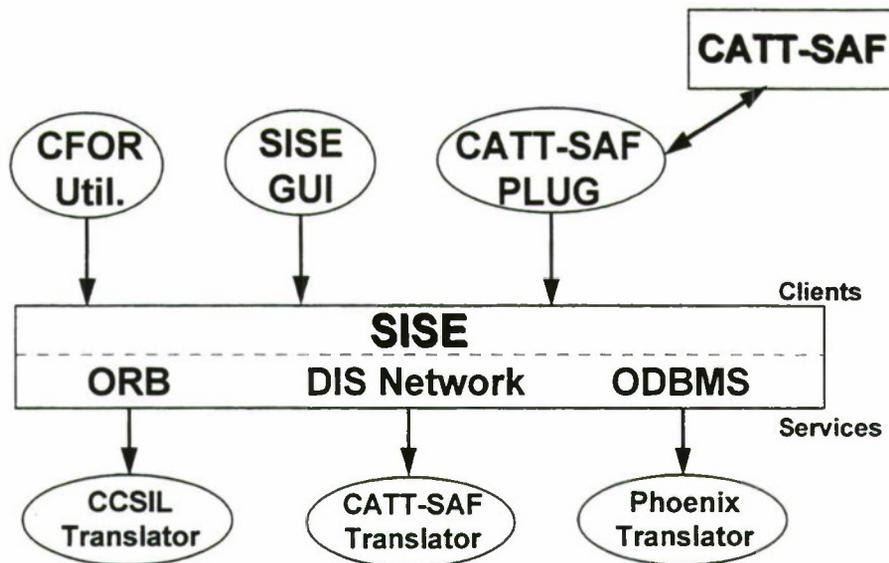


Figure 2: CFOR Linkage

The CATT-SAF plug used in this linkage is identical to that of the Phoenix to CATT-SAF linkage. Thus, the CATT-SAF plug communicates with the ORB and with CATT-SAF in the same manner as in the previous experiment. This means that the source of the order is transparent to the CATT-SAF end.

6. Results and Conclusion

These experiments were performed with Phoenix, the SISE GUI (used as a CFOR commander), and CATT-SAF executing simultaneously. This allowed orders to be sent from both Phoenix and the SISE-GUI to the same CATT-SAF exercise. The verification methodology for the experiments consisted of visually inspecting, after each order was sent, the changes to the CATT-SAF execution matrix and to the entities on the CCTT SAF Plan View Display.

Through these two experiments, we have demonstrated both interoperability and reusability of simulation components. Interoperability has been demonstrated at the command and control level across several different simulations. This was accomplished with minimal effort due to the common architecture design shared by all components and by the use of ORB technology.

In theory, a representation of the CCSIL standard could be the generic command and control format used in this architecture. A full linkage of CATT-SAF to CCSIL would allow a wide variety of

doctrinal command and control capability to be utilized among CFOR simulations while providing the platform interactions that are normally allowed via DIS PDUs. This will be accomplished through a planned linkage to ODBMS middleware such as Object Design's ObjectStore which will allow the storage of orders in a generic format resembling an object-oriented version of CCSIL.

7. References

- Kuhl, Frederick. DIS and Object Request Broker. In Proceedings of the 10th Workshop on Standards for the Interoperability of Distributed Simulations. Orlando, FL (1994), 43-46.
- Lander, W. B. Object-oriented technologies in SimCore. In Object Oriented Simulation, 1995. Proceedings of the 1995 Western MultiConference, 1995.
- Object Management Group (OMG). The Common Object Request Broker Architecture (CORBA) Specification. (1995).
- Peck, Dr. Charles C.; et. al. Applications of distributed object technology to DIS. In Proceedings of the 12th Workshop on Standards for the Interoperability of Distributed Simulations. Orlando, FL (1995), 537-552.

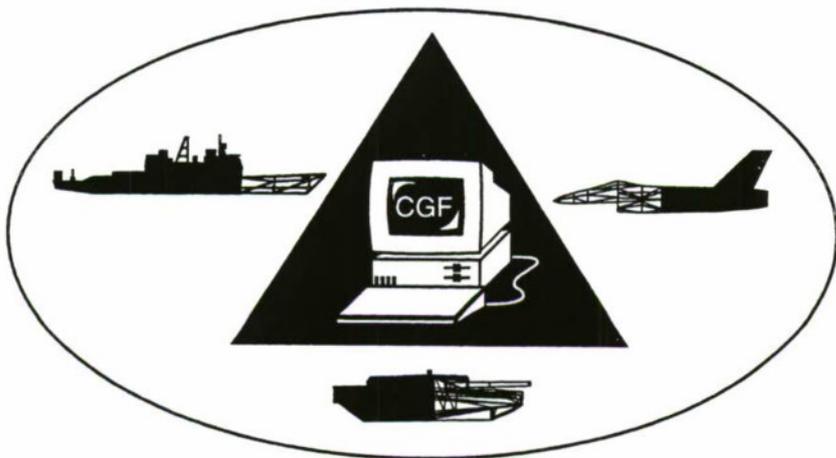
Sureshchandran, S.; et. al. Use of object request broker for dynamic environments in Distributed Interactive Simulation. In Proceedings of the 13th Workshop on Standards for the Interoperability of Distributed Simulations. Volume 1. Orlando, FL (1995), 65-67.

8. Authors' Biographies

Dr. Frederic (Rick) McKenzie has been a member of the Advanced Distributed Simulation Research Team (ADS RT) since April 1995 serving as P.I. for two interoperability IRAD projects. He holds a Senior Scientist position at SAIC Orlando and is currently technical lead on an ARPA sponsored advanced interoperability project. For two years prior to joining the ADS RT, he had been a member of the knowledge engineering team for the SAF component of the Close Combat Tactical Trainer (CCTT) project. Dr. McKenzie has had two years teaching experience in software languages and data structures. He obtained a Master of Science in Computer Engineering in 1990 and a Ph.D. in Engineering in 1994 from the University of Central Florida. Both his Masters and Ph.D. work have been in AI research, focusing on knowledge representation and model-based diagnostic reasoning.

Gregory Shumaker has been a member of the Advanced Distributed Simulation Research Team(ADS RT) since July 1994. He has been involved with several R&D efforts and is currently supporting an ARPA sponsored advanced interoperability project involving the verification and validation of SAF behaviors. Gregory has a Bachelor of Science in Computer Science from the University of Central Florida in 1993. He received his Master of Science in Computer Science in May 1996 from the University of Central Florida. His Masters work involved the Bulk-Synchronous Parallel(BSP) model of general purpose parallel computing.

Pete Campbell was a member of the Advanced Distributed Simulation Research Team (ADS RT) from October 1995 to May 1996. He is currently pursuing other goals. Pete has a Bachelors of Science in Computer Engineering from the University of Central Florida in May 1996.



Drilling CGF Agents in METT-T: an alternative approach to conventional AI

R.W. Penney
Defence Research Agency,
St. Andrews Road, Great Malvern,
Worcestershire WR14 3PS, U.K.
email: penney@signal.dra.hmg.gb

Abstract

While Western military doctrine stipulates that officers at all levels must weigh-together many disparate factors in their decision-making, whether reacting or planning, it is well-known that existing techniques for simulating such battlefield decision-making are generally rather poor at balancing these various disparate influences. Therefore it is not entirely certain whether even a semi-automated force having close human supervision of small groups of computer-controlled entities, could provide a realistically challenging opposition for individual humans. This is of particular concern in training or mission-rehearsal contexts if the low-level reasoning of individual simulated entities is so simplistic and inferior to that of the humans with which they must contend. We have therefore sought to develop algorithmic techniques that are better able to support rational artificial decision-making in complex non-stereotyped situations. An implemented application of these techniques to the versatile control of small numbers of simulated tanks moving in rich randomly-generated topography, without relying on numerous inflexible rules, will also be discussed. Thereby we will indicate how our algorithm is inherently able to balance the requirements of route-planning, enemy avoidance and force cohesion as circumstances dictate.

© British Crown Copyright 1996/DERA. Published with the permission of the controller of Her Britannic Majesty's Stationery Office.

1. Introduction

The impressive recent developments in creating virtual environments for military applications do much to support the substantial replacement of highly costly large-scale field exercises

by simulator-based alternatives. Even considering the, often underestimated, computational expense of transcribing a plethora of real-world physical phenomena (e.g. vehicle dynamics, inter-particle collisions, wind effects, and visual and sonic phenomena) into a computer model, the benefits of versatility, controllability and accessibility are quite persuasive. However, as the scale of exercises becomes greater, the extent to which training must be focussed on a relatively tiny set of the participants in order to be effective, becomes very marked. Clearly, the interest in minimizing the staffing requirements of such exercises by replacing as many, purely ancillary, staff by computer-generated forces (CGFs) is understandable, but makes extreme demands of the artificial forces if they are to be able to interact closely with humans practising and polishing their own skills.

1.1. Fidelity and Verisimilitude

It may seem that the many simulation systems used for operational analysis must provide some important universal paradigms in battlefield modelling. However, this would be to ignore some relevant general principles that are highly pertinent to the successes of modelling in general. Perhaps foremost amongst this are the concepts of universality and self-averaging (Fischer and Hertz 1991), whereby as the scale at which a system is observed increases, the sensitivity to the precise characteristics of the lowest-level dynamical laws and microscopic details diminishes. Hence, to a large extent the usefulness and temptation of such simulations rests on the ability of relatively crude modelling of entity-level physics to yet provide plausible *macroscopic* force behaviour. So, provided one is careful how one analyses the results from such systems (taking due account of random fluctuations,

and the depth to which the results can be trusted) the occasional failings and fortuitousnesses of their simplistic lowest-level behaviours may well cancel out on average. This means that producing statistically-correct large-scale models of collective behaviour can be greatly facilitated by exploiting this tolerance in the fidelity required at the lowest level of detail, even though this lower-level modelling is often the most convenient way of modelling the effective behaviour of larger composites (Richardson 1994). Furthermore, one should recognize that it is often very difficult even to determine whether individual entities are making sensible decisions when their behaviour is only observable on a plan-view display. Again, the larger-scale plausibility of force behaviour, when seen in plan-view, does not mean that individual entities are acting in a way that a human, in a similar position, would not consider crass.

Even though, when carefully used, relatively crude battlefield models can be generally instructive, it would clearly be cavalier to assume that such techniques are necessarily sufficiently accurate in their low-level details to be able to support human participation. The great fluency with which humans adapt to, and exploit, the peculiarities of a situation means that, far from being nullified on average by other random factors, the human's skills could potentially undermine the credibility of the simulation as a whole. For example, a failure of computer-generated tanks to be able to hide behind ridge-lines or manoeuvre collectively while covered from a moving enemy, would give human tank commanders an anomalously high kill-rate against such an opposition, that could drastically skew the outcome of an exercise. Indeed, such weaknesses are not simply avoided by the semi-automated force (SAF), unless their human controller has access to entity-level perceptions. Given the many conflicting military factors, such as Mission, Enemy, Time, Terrain, Troop (METT-T), that humans recognize as important to effective operations, these must surely be reflected in the decision-making of artificial forces if they are to form an effective opposition. Moreover, however facile *all* humans find spatio-temporal reasoning tasks such as hiding and moulding formations to context, it would be extremely naïve to believe that these abilities can be emulated without highly computationally-intensive calculations.

Indeed, this human ability is primarily a reflection of the wealth of intuition and common-sense that *underpins* domain-specific expertise; hence the emphasis on using doctrinal rules as the *foundation* for CGF systems is rather questionable if any real flexibility is expected. Certainly it is possible to regulate exercises and predetermine some of the likely circumstances within it, but given the growing emphasis on initiative and creativity in military doctrine it seems unrealistic not to expect, at least significant variations on the anticipated courses of action, to occur. Individual CGF entities have to be able to adapt to these variations autonomously, if they are to be of any real value in reducing staffing requirements in synthetic environments. Yet in order to adapt reliably to variations and combinations of stereotyped situations, it is not sufficient for agents to treat conflicting situational influences separately; more rational compromises must be found.

1.2. Divide and Contort

If one measures existing CGF technologies against these requirements, there are grounds for concern (e.g., Meliza and Vaden 1995), however impressive the progress that they may represent. In general the weakness of existing systems can be largely attributed to the discretization and compartmentalization that are so pre-eminent. For example, it is commonplace to find route-planning to be considered separately from mission-planning¹ and from formation-keeping, while clearly these problems are only independent in the most sterile of situations (c.f. Campbell *et al* 1995). There is indeed a tendency for unit behaviour to be generated as an isolated whole, without considering how this should be influenced by the behaviour of other units, or the implications for the constituents of single units (Ourston *et al* 1995, Harmon *et al* 1994). For example, planning a concealed route for an atomic tank troop does nothing to prevent the individual tanks straying out from behind ridge-lines through taking naïve action to keep in formation; an issue that may or may not have wider implications according to context. Similarly the style of Combat Instruc-

¹This is particularly relevant as route-planning techniques seem focussed solely around finding routes between *pre-specified* endpoints, while taking scant account of the *time-dependence* in the extraneous factors influencing the choice of route.

tion Sets (Ourston *et al* 1995) almost forces one into a finite-state-machine (Dougherty and Giardina 1988) formulation of behavioural modelling. For example, for a tank troop responding to indirect fire, it may have to commit either to hastily retracing its steps or to scattering into nearby covered positions, or to continuing with the mission regardless. However, even though all these options sound sensible, to assert that they are exhaustive, can all be clearly distinguished, and are each unambiguous, is somewhat unrealistic — there could manifestly be situations where a small change in the intended route ahead would allow the troop to continue with the mission while having the benefits of cover. While this slight generalization of a basic principle is utterly obvious to a human, too naïve a transcription of the basic set of options into software would totally prevent natural generalization when circumstances represent a mixture of the stereotyped cases. Moreover, merely deciding which of a limited set of options to adopt must be dependent on how these options will actually be enacted, and yet such information, as well as general contextual factors, are extremely difficult to include in this style of formulation.

In view of the ramifications of the apparently inevitable limitations of some of the more popular CGF techniques, we have sought to develop a novel formulation of artificial decision-making that avoids some of these weaknesses. In view of our scepticism of compartmentalization of behaviour, utter dependence on situation-specific expert knowledge, and too crude entity-level behaviour, we have aimed for a unified and intrinsically flexible entity-level planning algorithm. In the next sections we will indicate how a functional optimization process can be used to generate plausible entity-level behaviour for a small number of tanks moving in rich terrain features, when provided with only quite rudimentary knowledge. Given that (time-dependent) route-planning, force cohesion and enemy avoidance are thereby emergent phenomena of a single process, we will indicate how our system is able to respond to quite a broad range of situations without relying on numerous prescriptive rules, and when given only the most limited of prescribed knowledge.

2. Conceptual Overview

Absolutely central to flexible behaviour generation, in any autonomous agent, is the ability to make *rational* choices between possible actions, and rational compromises between competing actions. This inevitably requires that agents have sufficient information at their disposal to be able to make these choices; there is no sense in expecting a context-sensitive decision to be achieved if an agent has an inadequate picture of its circumstances. Although some popular CGF technologies may have great strengths in the lucidity or speed of their reasoning systems, and have functionality that is *analogous* to human military thinking, when analysed more closely it is clear that they typically *inherently* base their entities' behaviours on far narrower information than is important to human adaptability. Most notably, they often expect to make decisions based on single snapshots of a battlefield situation, or to have semi-automated entities execute their operator's orders without any idea what factors lay behind his choice of those orders.

We have already mentioned the importance of intuition in fitting expert knowledge into context, but it should be apparent that the doctrinally-based branch-points in the pandemic rule-based systems or finite-state machine architectures simply *decree* certain actions be taken; it thereafter being the responsibility of each triggered action to adapt, alone, to the prevailing situation. However, in complex environments it is optimistic in the extreme to assume that the outcome of a course of action can, with any generality, be reliably determined without some form of simulation, whether performed computationally, mathematically or cerebrally. If it is not realistic to predict the outcome of a course of action without a simulation, then it is surely unrealistic to determine which of a set of courses of action to adopt without some similarly elaborate evaluation of each of them. Yet this appears to be exactly what Combat Instruction Sets, Finite-State Machines and rule-based systems are attempting to do. For example, determining whether one should instruct a subordinate to defend a particular corridor of retreat must depend on one's expectations about how a preceding assault may develop over time, and the likelihood that that particular mode of retreat will be helpful. It would be a very crude assumption to

assert that there is any *intrinsic* value to sending out orders that an arbitrary manoeuvre pathway should be defended. Moreover, pre-processing a pre-scripted exercise to identify 'strategically important' corridors seems anathema to flexibility and to the idea that a CGF system can be a challenging, yet labour-saving, opposition to a human force.

While directly useful in strongly stereotyped situations, doctrinal knowledge itself should be seen as reducible to, and justifiable in terms of, some underlying principles. Put crudely, one might say that the simplest principle was 'avoid death', with expert wisdom providing good suggestions of how to achieve this in particular circumstances, given known mechanical properties of vehicles, terrain etc. As such, the principles *behind* doctrinal knowledge provide not only a way of validating it against a given context, but also equip one to deal with novel situations to which no familiar maxim really applies. So it is helpful to see doctrine as a set of suggestions, that when evaluated can often be seen to be viable, rather than as a set of prescriptions. Indeed, as suggestions, they should also be seen as mere sketches that may be combined with existing strategies, and which will inevitably need fitting into context on the basis of the basic principles of which the doctrine is a heuristic assemblage.

Having already noted that flexibly making rational context-dependent decisions must rely on some form of mental extrapolation of the current situation, it is clear that intelligent behaviour cannot simply be a process of continually deciding what to do next — the consequences of actions may take time to emerge, may obviously be affected by actions one takes in the interim, and could well be influenced by the activities of other entities. Moreover, it is seldom the case that the effects of actions are confined to instants in time — for example, the act of shooting an enemy affects the viability of the entire future period that this enemy could have posed a threat. Similarly, fuel costs, or the exposure suffered during an advance, accrue over time rather than at isolated instances. Therefore, the process of evaluating a plan is not easily reduced to a simple evaluation of an end-state, rather it involves an assessment of the *entire* history of that plan, taking due account of the time dependences

in the exercise beyond one's own control.

To summarize, the principles we see as fundamental to achieving flexible autonomous agents for complex environments such as virtual battlefields, are as follows:

- the use of low-level knowledge as the discriminant for all decisions;
- doctrine being used to provide *suggestions* of advantageous behaviour, to be evaluated against low-level understanding;
- the use of mental simulation to assess an intended course of action in its effects over time;
- an intrinsic mechanism for exploring variations on a course of action, which may be used to adapt doctrine to context, or to generate new behaviour;
- the unification of all decision making into one mechanism, so that compromises between competing aspirations can be achieved *rationally*, rather than by decree.

3. Analytic Formulation

Having highlighted the importance of intuition to human adaptability, and the inescapable computational expense of implementing skills that almost any human would consider trivial, it is hazardous to formalize an artificial decision-making system through only a linguistic description (c.f. Hieb *et al* 1995). However plausible such a description may appear, when interpreted by a human undue allowance for its omissions, assumptions, and ambiguity may well be perpetrated. Given that neither computer hardware, nor electronic hardware in general, is fundamentally capable of performing more than purely arithmetic operations, it behoves one to formulate one's problem in similar mathematical terms, if one is not to be seduced by mere analogies between one's expectations and what is *genuinely* implementable².

3.1. Cost-functions and basic knowledge

With this need for a fairly reductionistic formulation of autonomous-agent behaviour in mind, we

²Indeed, part of the magical power of human intelligence is this ability to see analogies between rather different things, yet this makes modelling intelligence difficult even to confront seriously.

now turn to a brief specification of how our agents choose their behaviour. Fundamental to this is a cost-function \mathcal{L} that is to embody the low-level knowledge that we see as central to flexibility. Although this cost-function is to provide a means of assessing each instant within the course of an exercise against a given entity's aspirations, this by itself is not enough to allow rational choices of courses of action. The importance of visualizing the consequences of each entity's actions over time, i.e. mental simulation, means that the viability of a course of action (starting at t_0) is determined not by \mathcal{L} itself but by a time integral

$$S = \int_{t_0}^{\infty} \mathcal{L} dt. \quad (1)$$

(S is therefore a functional of the future course of events.) Given that this is a process of *mental* simulation, it must be an artifice of a single entity's mind, and hence involve not only that entity's intended activities, but also hypotheses about how other entities will act. (The rich geometry of terrain alone means that mathematically non-trivial hypotheses will be necessary even to allow extrapolation of entities' trajectories around rudimentary terrain features.)

As a simple example, suppose that in a particular entity i 's mind, there are a set of phantom entities $\{j = 1 \dots N\}$ (one of which will represent the physical entity i itself), and that each of these phantom entities has a hypothetical trajectory $\vec{r}_j(t)$ in the xy -plane. Given a terrain surface $z(\vec{r})$, there will be a term in the cost-function due to the rate at which entity i consumes fuel;

$$\mathcal{L}_{fuel} \left\{ \sqrt{(\dot{\vec{r}}_i)^2 + (\dot{z}_i)^2} + g\Theta(\dot{z}_i) \right\}. \quad (2)$$

wherein

$$\Theta(x) = \begin{cases} x & x > 0 \\ 0 & x \leq 0 \end{cases} \quad \text{and} \quad \dot{z}_i = \dot{\vec{r}}_i \cdot \nabla z(\vec{r}_i). \quad (3)$$

When integrated over time, this part of the cost-function represents the total fuel consumed during the vehicle's motion, through work done against friction and gravity by motion up gradients. It is therefore clearly dependent on the topography traversed and not solely determined by the end-points of the route followed.

Similarly, one could discourage entities from moving too close to enemies by a term in \mathcal{L} of the form

$$\mathcal{L}_{death} \sum_j \nu_j \exp\left(-\frac{|\vec{r}_i - \vec{r}_j|}{\lambda_{ij}}\right), \quad (4)$$

in which ν_j is the notional firing-rate of entity j , and λ_{ij} is a representative lethal range (perhaps depending on the existence of line-of-sight between entities i and j). Here again, the viability of a particular route $\vec{r}_i(t)$ is not solely a reflection of the exposure suffered at isolated instances, nor is it uninfluenced by the *motion* of the hostile forces. This term captures more flexible knowledge through taking account of the separation between entities, and being more clearly motivated in terms of a mechanical model of shell-firing, rather than enemy avoidance simply being treated as a constraint based on *fixed* enemy locations (c.f. Karr *et al* 1995, Longtin *et al* 1995).

One can also encourage formation keeping by including terms such as

$$\mathcal{L}_{group} \sum_j \exp\left(-\frac{|\vec{r}_i - \vec{r}_j|}{\xi_{ij}}\right) \left\{ \frac{r_0}{|\vec{r}_i - \vec{r}_j|} - 1 \right\}, \quad (5)$$

where ξ_{ij} represents that range of attraction between entities of the same force and r_0 reflects the minimum safe inter-vehicle separation. That this term only *encourages* friendly forces to remain together means that they are not simply tied together and incapable of acting independently or separating when situations favour this. Once again, the accumulation of the cost-function into a figure-of-merit S for an entire trajectory $\vec{r}_i(t)$, allows groups of entities to fragment when necessary while anticipating that they will re-group at a later stage. This seems a very necessary ability for groups of entities that must cooperate while performing differing roles, such as when executing a two-pronged attack.

3.2. Striking Compromises

Now, by simply adding together the component cost-functions (2, 4 & 5), and evaluating their time-integral (1), one has quite a rich assessment not only of the specifics of a course of action, but also its context. For example, the viability of re-joining one's colleagues can be assessed against the need to cross intervening terrain, and the expected

motion of enemy forces as well as any expected motion of the friendly forces. This clearly equips the entities to make a far more sensible judgement of whether re-joining their formation is actually advantageous, rather than blindly being attracted to a point in a template formation sited at a fixed location.

Similarly, because fuel-costs (2) and enemy-avoidance aspirations (4) are part of the same assessment process, there is automatically an incentive for entities to avoid enemies without crossing awkward terrain. Hence, enemy avoidance is not dependent on an entirely separate mechanism providing a protective location, to which a route is found by yet another mechanism (which generally will not take into account the motion of the enemy). In the present formulation, there is much more of a pliable compromise between adopting a safe retreat and conserving fuel³, and situations where an inaccessible covered location is selected, or where the nearest covered location is not the most appropriate, are less likely to occur.

Indeed, it seems quite reasonable that consideration of ways of avoiding an enemy force need not be restricted to simply taking up static covered positions. Moreover, when entities' plans are generalized to include hostile actions such as firing shells, entities are then equipped to make some rational choice between reducing such threats either by confronting or by avoiding them. When shooting is seen as a separate behaviour, perhaps through being purely reactive, then entities cannot easily adjust other aspects of their behaviour to make best use of their shooting abilities.

3.3. Plan Optimization

So far we have discussed some of the implications of a decision-making process based around (1), in terms its favouring certain sensible courses of action over others, especially in non-stereotyped situations. However, to be able to function autonomously, entities must be autonomously able to think of courses of action (or plans) prior to being able to make any rational choice amongst them. Hence, behaviour-generation is essentially an op-

³Although we will refer to 'fuel costs' as though they were solely due to some economic value, more generally they can capture related notions of general expediency while in transit.

timization problem, where rather than optimizing a simple function, it is a model of the future evolution of an exercise (1), that is the object being optimized.

Clearly the space of all possible plans is enormous and it would be absurd to expect to be able to find strictly optimal plans. Conventionally this is seen as a motivation not only for compromising the scrupulousness of the *search* for a good plan, but also for compromising the *evaluation* of all plans (c.f. Hoff *et al* 1995). Hence, in order to make (relatively) complete examination of the range of plans possible, the space of plans is often coarsened through some form of discretization that inevitably subverts the evaluation of the phenomena produced by each plan, and hence reduces context-sensitivity. However, as well as having weaknesses in terms of the crudity of the evaluation of a plan⁴, discretization is likely to obfuscate the broad range of lengthscales and timescales over which battlefield phenomena occur. For example, one would like it to be a fairly simple matter not only to consider small perturbations to a trajectory to better negotiate motion up a slope, but also to make comparison of alternative routes either side of a hill similarly natural. It is not easy to achieve both these benefits with an artificial discretization of the terrain (such as into a grid of vertices), while still preserving the ability to refine a plan as events unfold on the (real) battlefield. Nor, given what we have said earlier about the consequences of earlier decisions taking time to emerge, is it wise to consider committing to the early stages of a course of action without even the vaguest notion of how it may conclude.

Although it will be illustrative to consider the process of behaviour-generation as one of optimizing simply the trajectories $\vec{r}_j(t)$ of a set of entities (within a particular entity's mind), in general these should be thought of as metaphors for all their actions. Hence plans may include shooting events (which in general may affect the population of live entities or intact bridges etc.), communication events, and other actions that have observable influences on the development of an exercise. As such, perhaps together with other sources of time-

⁴There is also a tendency for an unrealistically short planning horizon to be chosen, or for the effects of a plan to cease when a particular objective is expected to have been met.

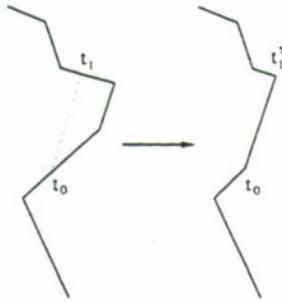


Figure 1: A schematic indication of the effect on a vehicle's trajectory of a simple transformation of the vertices, replacing part of the trajectory between t_0 and t_1 by its resultant.

dependence such as weather, they provide a means by which an entity can form a hypothesis about any instant in the future of the exercise⁵.

Given the status of a set of plans as a means of describing the anticipated evolution of an exercise, it is helpful to formulate them so that they can describe the state of the exercise at any time, even when the plans are in their most rudimentary incarnations. In addition, to reflect the range of lengthscales and timescales over which relevant phenomena may occur, plans should have an expressive form that allows alterations on various scales to be effected concisely. For example, one may consider a trajectory to consist of a series of line-segments, with the coordinates of the vertices being events within a plan. Simply by inserting or deleting events within such an event-list, or by altering the locations of the vertices, one can make natural distortions of the induced trajectory on whatever lengthscales is desired. Such a representation allows one to have certain sections of a plan refined in great detail while still allowing crude sections of a trajectory to be described concisely and in a form that allows the effects of following the entirety of such a trajectory to be examined. For example, figures 1 and 2 illustrate some representative transformations of a trajectory that form part of the plan-optimization process, and can be effected by simple adjustments to the events within a plan.

Given that it is not tenable to expect to find truly

⁵It would be highly desirable if one could confine attention solely to *relevant* moments in the hypothesised future, but this is highly non-trivial to achieve with worthwhile flexibility in domains of any complexity.

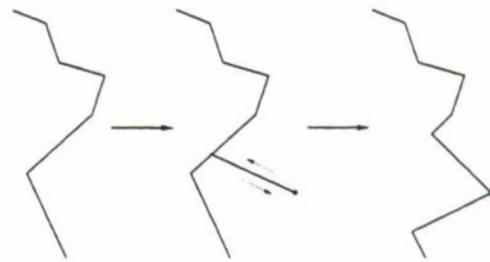


Figure 2: Illustration of a series of trajectory-transformations that together add a detour to the initial route.

optimal plans, and our reluctance to sacrifice the *evaluation* of a candidate plan, it is natural to use some form of stochastic optimization process to refine entities' plans. For example, simulated annealing may provide an effective means of affordably exploring a representative sample of the space of all possible plans. Just as complete, deterministic, search methods rely on a judicious choice of the order in which the points in the search space are examined (for example Gray-coding or A* search), stochastic optimization relies on a careful choice of a dynamics on the search space. Conveniently, a set operators that produce simple deformations of a plan can be used as the basis of such a dynamics. Hence, the style of distortions shown in figures 1 and 2, together with simple rotations, reflections and addition or removal of shooting events, form a major part of the means by which effective plan-refinement can occur. Moreover, because the character of these operators is dictated primarily by their geometrical appeal, rather than the representation used to describe plans, they can together operate on a wide range of lengthscales and timescales, rather than all always having to make the smallest changes to a plan.

Given such a set of plan-adjustment operators, which provide a stochastic dynamics on the space of plans, plan optimization proceeds by continually making tentative adjustments to a plan, evaluating the cost-functional of the new plan (1), and accepting the alteration according to the change in cost-functional. In simulated annealing this is done on the basis of favouring moves to plans of lower cost, but tolerating moves that increase the cost by amounts similar to the scale provided by the annealing temperature. By progressively reducing this temperature, from an initially warm level, the intention is that more subtle refine-

ments to a plan are made generally only after its grossest features have been established. There is also the considerable advantage that the optimization is not absolutely dependent on the environment remaining constant as the optimization is in progress, because the process centres around comparisons between only pairs of plans, rather than expecting to be able to refine the later stages of a plan without its earlier stages being invalidated as events unfold. This offers the possibility of allowing entities to continually refine their plans concurrently with acting upon them, without having either to think only once for each set of orders or to have to stop and deliberate whenever circumstances change. While a convenient paradigm with which to compare our actual optimization process, there are nevertheless some significant differences from classical simulated annealing, which we will not discuss here. (Further details may be found in Penney 1996.)

Two further roles for the operators at the basis of the stochastic dynamics, are worth alluding to. The first relates to the fact that the battlefield situation in which each entity is refining its own plan is constantly evolving, both through making observations and also through developing its hypotheses about how other entities will act. Hence, it is extremely helpful if entities are able to make rapid initial refinements to their plans in response to changing circumstances, before more considered changes can occur. The human ability to be able to visualize bending trajectories around terrain features, or deforming these to keep a set of trajectories closer together, is one that is extremely powerful. Something of this ability can be captured by including amongst the stochastic plan-dynamics an operator that uses gradient information within the space of plans. For example, if a plan is described by a set of parameters ξ^α , then the gradients $\partial S/\partial \xi^\alpha$ provide suggestions how to adjust each parameter so as to reduce the cost-functional S . Moreover, by careful choice of the representation used to describe plans, one can conveniently exploit the expression

$$\frac{\partial S}{\partial \xi^\alpha} = \int_{t_0}^{\infty} \frac{\partial \mathcal{L}}{\partial \vec{r}} \cdot \frac{\partial \vec{r}}{\partial \xi^\alpha} + \frac{\partial \mathcal{L}}{\partial \dot{\vec{r}}} \cdot \frac{\partial \dot{\vec{r}}}{\partial \xi^\alpha} + \dots dt. \quad (6)$$

to allow efficient calculation of these derivatives concurrently with the same mental simulation used to evaluate the plan itself.

The second additional aspect of the plan-refinement operators is in supporting expert knowledge without this being prescriptive. As the conditions on the set of operators are relatively weak (such as an aspiration for ergodicity (Reichl 1980)), it is quite legitimate to extend these operators to allow distortions of a plan in accordance with strategies or behaviours known to be generally beneficial. For example, entities can be encouraged to consider keeping in formation through an operator that suggests distortions to their trajectory such as to bring parts of it closer to those of their colleagues. However, because this advice is given through mechanisms that as subjected to the same plan-evaluation as are tentative adjustments that are made purely at random, it is not treated as infallible advice, and can be judged against the prevailing context. Once accepted, even if representing only relatively crude advice, the effects of expert-operators can then be better fitted into context by the basic stochastic plan-dynamics that are central to the general efficacy of the optimization.

4. Simulation Results

Now that we have given a brief overview of the way in which our entities' behaviour is generated, and the motivation for this, we can now discuss some of the phenomena this mechanism actually produces. In order to test whether the process exhibits the flexibility desired, and is able to behave rationally in a complex environment without depending on copious human-supplied rules, we have constructed a rarefied military scenario on a randomly-generated terrain surface. By insisting that our entities are able to contend with a terrain that is realistically undulating, and is not artificially easily debased into a collection of simple hills and valleys, their 'understanding' of basic concepts in spatio-temporal reasoning is being much more powerfully tested than by a scenario always conducted on an unvarying terrain. Together with the need to reason about the motion and actions of both friendly and hostile forces, the chosen scenario captures some of the more demanding characteristics of an exercise subject to human participation, especially its variability and only loose dependence on template terrain features.

With the profile of a fresh instance of the terrain known to all entities (to avoid having to update

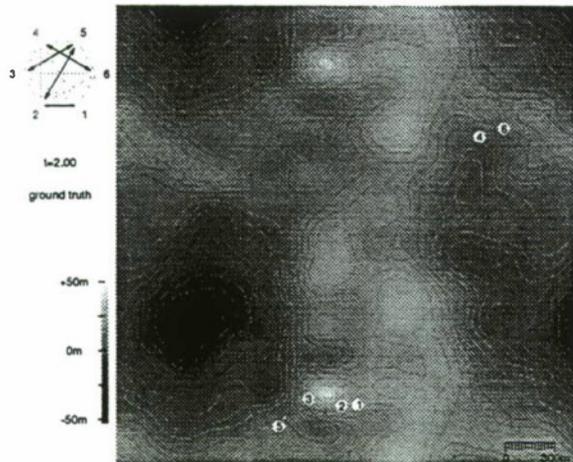


Figure 3: An indication of ground-truth two seconds into a simulation. Terrain contours are shown at 2.5m intervals, with line-of-sight between vehicles shown in the hexagonal icon (arrow-headed solid lines indicating the existence of line-of-sight, which in general need not be symmetric). Shells are marked as short thin two-tone lines, with the white part showing the direction of travel.

their beliefs about such a geometrically subtle object), the scenario opens (figure 3) with a force of three (blue-force) tanks 1-3 towards the south, with an opposition of three (red-force) tanks 4-6 split between the north-east and south-west. Once again, this scenario is chosen in the interests of exposing the entities to challenging situations *qualitatively* similar to the more demanding aspects of real military exercises, rather than being intended to be militarily realistic in a more ostentatious sense. The tanks have varying aspirations, embodied by their respective cost-functions, but all are concerned with conserving fuel, avoiding live enemies (perhaps by shooting them), and remaining close to friendly forces. The opening positions of the entities means that tanks 2 and 3 have each already seen tank 5, and decided to opened fire. (Here shells follow parabolic trajectories and have finite times of slight, which the firers attempt to allow for when shooting at a moving target.) At this time, tank 1 is still unaware of tank 5, and also of its colleagues' actions against this opponent. All the blue tanks are oblivious to the existence of the other red tanks to the north-east.

By twenty seconds into the simulation, tank 4's picture of the battlefield situation is plausible, but not wholly accurate (figure 4). Before its colleague, tank 5, was destroyed by $t = 4s$, the latter

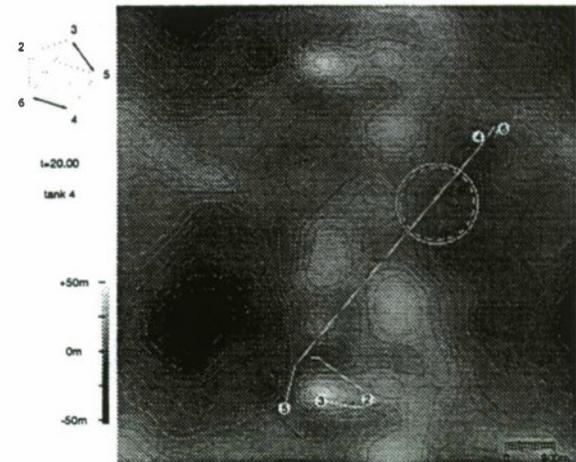


Figure 4: (Red) tank 4's mental image of the exercise, after twenty seconds. Vehicles' anticipated trajectories are shown as thin white lines emerging from them. The full circular line denotes tank 4's belief that the blue force will try to attack the region (shown by the dotted circular line) that it is to defend.

had communicated its sightings of tanks 2 and 3 (tank 1 having been occluded by the terrain). The figure shows that tank 4 still believed tank 5 to be alive, that the latter would proceed north-easterly to join it and tank 6, and that tanks 2 and 3 would remain to the south. (It is not clear whether it thought that tank 3 would move to attack tank 5 while *en route*, or whether it was expecting to seek cover in a depression after tank 5 had moved further away.) The defensive goal shared by all red tanks is also shown as the dotted circle in figure 4, and they all assume that any blue tanks have an incentive to assault this region, as indicated by the solid circular line in the figure. At this time, tank 6 has also started to form hypotheses about the intentions of the other tanks, but although qualitatively similar, this tank presumes that tank 5 will proceed east before turning northerly to join it and tank 4, while the two blue tanks of which it is aware retreat northerly.

In figure 5, tank 1's mental picture of reality is shown for thirty seconds into the exercise. This tank has been given two successive *rendez-vous* goals, shown as open circles. However, neither of its colleagues has any direct incentive to move through these regions; that they should do so at all is a reflection of their desire to remain close to their colleague. (This is intended to establish that entities are able to cooperate even when not per-

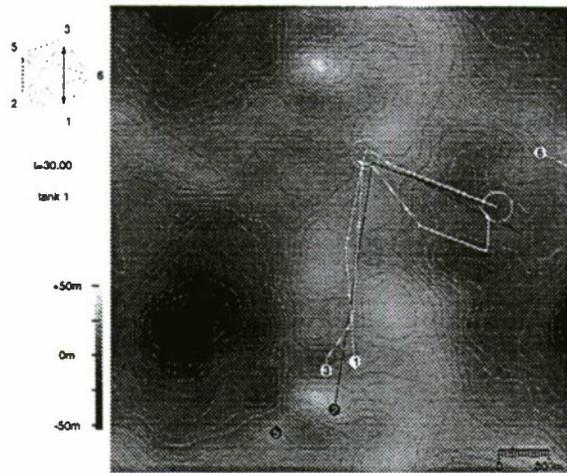


Figure 5: (Blue) tank 1's view of the exercise, after thirty seconds, by when tanks 2 and 5 are known to have been destroyed. Tank 1's two *rendez-vous* goals are shown as open circles.

forming identical functions; an important ability of any unit that must sub-divide and yet still act in coordination.) Tank 1 has clearly identified a sensible route through its two goals, while taking account of intervening terrain and the presence of tank 6 (which was first seen at about this time). Tank 1 also believes that tank 3 will take an alternative route, and one that keeps slightly further away from tank 6 while exposed in the valley to the east, but still rejoining tank 1 after a period out of sight of both it and tank 6. Indeed, tank 3's intended trajectory clearly represents a compromise between keeping close to tank 1 as it moves, minimizing fuel costs through attention to the terrain traffickability, and avoiding an enemy (which itself is expected to be in motion). Before tank 2 had been destroyed by tank 5 (at about $t = 4s$), tank 1 thought that it too would follow the rest of the blue force. Tank 6 was expected to retreat in the face of this strong presence.

Shortly before tank 1 reached his first *rendez-vous* position (figure 6), it and tank 3 had seen that there were in fact two red tanks to the north-east, and had revised their routes to take advantage of cover provided by the northerly range of mountains in the centre of the figure. (The discontinuity in tank 3's trajectory is particularly evident as tank 4 was sighted as they emerged from a mountain pass.) Believing the entire blue force still to be intact, and with tank 6 by now destroyed, tank 4 decided to retreat further north-east behind an-

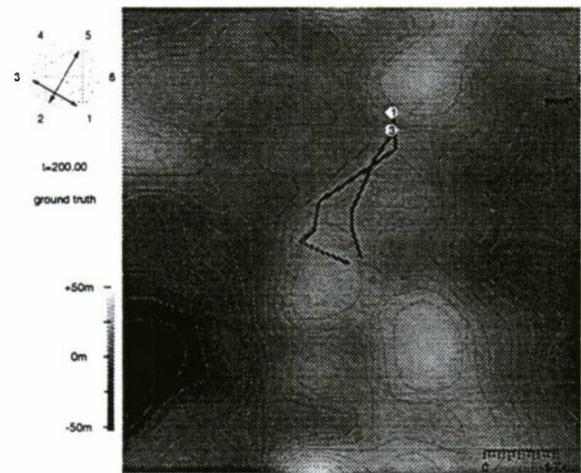


Figure 6: Ground truth after two hundred seconds. The black trails show tanks' motion during the last hundred seconds. Tank 4 has retreated to the east of the figure, while tank 6 has been killed.

other range of hills, leaving the two blue tanks to proceed to tank 1's second *rendez-vous* unhindered.

This simulation history, which although chosen for its illustrative merits is qualitatively similar to most other runs of the simulation, is fundamentally a reflection of very sparse knowledge that is supplied to the entities through their cost-functions. Given only a handful of terms in their respective cost-functions, together with a few expert operators (to assist in not wasting shells on dead targets, in keeping in formation, and in providing a vertex in their trajectory near a *rendez-vous* goal) they show some very encouraging adaptability. Indeed, they show pleasing response to the terrain, as both an obstacle and as cover, without any need for this to be pre-processed to identify 'significant' features.

While with careful construction the simulation runs only a few times slower than real-time on a relatively modest multi-processor workstation, and shows the techniques not to be impossibly expensive, it is clear that the price of such adaptability is not inconsiderable. Certainly there is much relevant cognitive functionality that is not supported within our current implementation, nor is likely to be affordable. However, one of the serious hazards of a more intuitive and less mathematical approach than we have advocated, is that the computational costs of rational adaptation to an ordinary range of

circumstances will be drastically underestimated. Hence, whilst pragmatism must play some part in the design of computer-generated forces in general, it is unlikely that the problem domain admits any expedient solution that does not have very serious pathologies when measured against humans' ability to 'muddle through' under a wealth of differing circumstances.

5. Conclusions

We have argued that, despite some impressive achievements, conventional CGF technologies tend to rely too heavily on an artificial segmentation of a strongly coupled problem, and on *intrinsically* imprecise linguistic descriptions of military decision-making. While such technologies can be capable, affordable, and accessible, their inherent weaknesses are often not appreciated. Indeed, it appears that segmentation of behaviour cannot help but undermine an ability to strike the compromises that are so important in complex environments, and that a conceptual description of human decision-making is not precise enough, and too reliant on an underlying intuition, to be directly transcribed into versatile computer software.

With these reservations in mind, we have constructed a unified decision-making system for individual CGF entities that is based on low-level knowledge. We have indicated how, through a process of mental simulation and an optimization process, this mechanism is intrinsically able to strike context-dependent compromises between competing aspirations, such as conserving fuel, avoiding enemies and remaining near friendly forces. So, rather than being separate areas of functionality, the classical behaviours of route-planning, enemy avoidance, and formation-keeping, are emergent phenomena of a single mechanism when viewed in particularly simplified situations. We have shown that this mechanism can be affordably implemented in quite a rich scenario, and indicated how quite capable behaviour of small numbers of tanks can emerge in a randomly-generated terrain surface without reliance either on pre-processing or on copious human-supplied rules.

Nevertheless, although our techniques are not unworkably expensive, through advocating a fairly mathematical formulation of CGF functionality, we have argued that the complex geometrical

and dynamical structure of military environments means that even semi-automated adaptable behaviour cannot be achieved reliably without considerable computational power. We have therefore sought to caution against overlooking the basis of military expertise upon intuition, and against believing that the ordinariness of such intuition, amongst humans, means that it is not exceedingly subtle to emulate on machines that can scarcely do arithmetic.

Acknowledgements

There have been a great many people who have been influential in the development of this work. Amongst these, we would particularly like to thank Jeremy Baxter, Mark Eaton, Mike Kirton, Brian Roberts, and Jake Turner, for their thoughts and encouragement at various times.

References

- Campbell C, Hull R, Root E and Jackson L 1995; "Route Planning in CCTT" in proc. 5th CGF&BR conf., pp233-243, Institute for Simulation and Training technical report IST-TR-95-04
- Dougherty E R and Giardina C R 1988; "Mathematical Methods for Artificial Intelligence and Autonomous Systems" Prentice-Hall
- Fischer K H and Hertz J A 1991; "Spin Glasses" Cambridge University Press (Cambridge)
- Harmon S Y, Yang S C and Tseng D Y 1994; "Command and Control Simulation for Computer Generated Forces" in proc. 4th CGF&BR conf., pp263-273, Institute for Simulation and Training technical report IST-TR-94-12
- Hieb M R, Tecuci G, Pullen J M, Ceranowicz A and Hille D 1995; "A Methodology and Tool for Constructing Adaptive Command Agents for Computer Generated Forces" in proc. 5th CGF&BR conf., pp135-146, Institute for Simulation and Training technical report IST-TR-95-04
- Hoff B, Howard M D and Tseng D Y 1995; "Path Planning with Terrain Utilization in ModSAF" in proc. 5th CGF&BR conf., pp255-263, In-

stitute for Simulation and Training technical report IST-TR-95-04

Karr C R, Rajput S, Cisneros J E and Nee H-L 1995; "*Automated Mission Planning in ModSAF*" in proc. 5th CGF&BR conf., pp159-168, Institute for Simulation and Training technical report IST-TR-95-04

Longtin M 1995; "*Concealed Routes in ModSAF*" in proc. 5th CGF&BR conf., pp305-313, Institute for Simulation and Training technical report IST-TR-95-04

Meliza L L and Vaden E A 1995; "*Measuring Entity and Group Behaviors of Semi-Automated Forces*" in proc. 5th CGF&BR conf., pp181-192, Institute for Simulation and Training technical report IST-TR-95-04

Ourston D, Blanchard D, Chandler E and Loh E 1995; "*From CIS to Software*" in proc. 5th CGF&BR conf., pp275-285, Institute for Simulation and Training technical report IST-TR-95-04

Penney R W 1996; "*A Statistical Physics Approach to Artificial Intelligence for Complex Virtual Worlds*" (unpublished) Defence Research Agency technical report, to be submitted to Proc. Roy. Soc. A

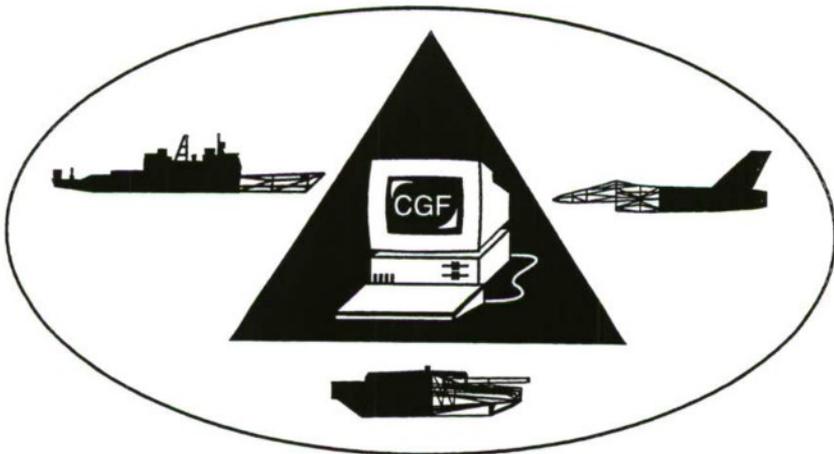
Reichl L E 1980; "*A Modern Course in Statistical Physics*" Arnold

Richardson S B 1994; "*Modelling of Adaptable C3I Systems (final report)*" (unpublished) Defence Research Agency technical report

Richard Penney read Physics at Oxford University, where he also gained his D.Phil. in Theoretical Physics on 'The Statistical Mechanics of Neural Networks and Spin Glasses'. Now working for the U.K. Defence Research Agency on Autonomous Command and Control Agents, his research interests remain founded on statistical mechanics, stochastic processes, optimization problems and mathematical modelling.

Session 8a: Terrain Modeling

Stanzione, TASC
Reece, UCF/IST
Stanzione, TASC
Stanzione, TASC



Multiple Elevation Structures in the Improved Computer Generated Forces Terrain Database

Thomas Stanzione
Forrest Chamberlain
Larry Mabijs
Mike Sousa

TASC
55 Walkers Brook Drive
Reading, MA 01867
tstanzione@tasc.com

Dr. Alan Evans
Cedric Buettner
Jonathan Fisher
Howard Lu

SAIC
Suite 130
20 Burlington Mall Road
Burlington, MA 01803
aevans@bos.saic.com

1. Abstract

The Improved Computer Generated Forces Terrain Database (ICTDB) project, a joint effort between TASC and SAIC, is one of the four ARPA/TEC Advanced Distributed Simulation Synthetic Environments projects. Its goal is to design and implement the next generation terrain database representation for Computer Generated Forces (CGF) systems. One of the areas that ICTDB is focusing on is the representation of multiple elevation surfaces. These include features with multiple elevations that are integrated with the terrain surface, such as bridges, tunnels, caves, and building interiors with multiple floors. These features are necessary to allow ModSAF vehicles to traverse multiple elevation terrain, such as highway overpasses and underpasses, drive into buildings, caves, and tunnels for concealment and other purposes, and allow individual combatants to move within buildings, which is especially important for Military Operations in Urban Terrain (MOUT).

In this paper, we describe the extensions that the ICTDB project has made to ModSAF in order to provide support for these multiple elevation surfaces. A new volume feature type has been added which represents these multiple elevation surface (MES) structures as both abstract features and full three dimensional models. MES structures are represented with enclosures and apertures. Enclosures are defined by solid surfaces used to represent walls, floors, and ceilings. Apertures are the areas that allow movement between enclosures, and include door and window openings, cave and tunnel entrances, and holes caused by battle damage or demolition. Topological information between enclosures is provided for movement and planning purposes. A three

dimensional virtual grid was implemented to spatially organize the polygons that represent the surfaces and apertures. ModSAF terrain algorithms, including elevation lookup and intervisibility, have been modified to work with these MES structures. For example, a new multiple elevation lookup function has been developed which returns a list of all surfaces and their elevations and surface types, which can be used by the place entity function to explicitly place an entity on a specific surface.

2. Design

2.1 Overview

One of the tasks of the ICTDB project is to expand ModSAF to represent multi-level buildings, bridges, caves, and tunnels. Because there are similarities in the geometry and line of sight characteristics of all of these types of features, a unified representation was created for any multi-elevation surface structure. The libCTDB functionality was extended so that vehicle placement, intervisibility, and elevation lookup calculations work within multi-elevation surface structures as well as terrain.

Multiple elevation surface (MES) structures are represented as a new volume feature subclass in libCTDB. These features contain a roof outline and a reference to an MES data structure. The data structure for an MES structure consists of a header, a list of enclosures, a list of apertures, a list of triangles, and a hierarchy of grid-boxes.

The header specifies the origin of a local coordinate system for the MES, in the Global Coordinate

System (GCS) developed earlier in this program (Evans, 1995). The origin can be chosen to be in any relation to the MES, i.e. it is not required to be at the southwest corner. The header also specifies a transformation matrix from the "world" space of the terrain database containing the MES to the MES space. This allows the creator of the MES to generate all information in terms of a local coordinate system. The origin and transformation matrix can be used to translate between MES local space and the world space. Modification of the origin and transformation matrix also allows the simple reuse and relocation of any MES structure, allowing MES structures to be instantiated in multiple locations on a database. Finally, the header also specifies a fixed point basis (meters/unit), which is the unit in which all coordinates in the MES are stored. MES structures utilize a grid box organizational structure, which is used for efficiency in the algorithms that operate on them.

2.2 Enclosures

An MES structure contains a set of "enclosures". As its name suggests, an enclosure is essentially an enclosed volume. Enclosures are created so that their contents are logically grouped together. An enclosure can be convex or concave, but must be a single level. That is, along each vertical line which intersects the enclosure, there may be at most one positive normal surface (e.g. a floor) and at most one negative normal surface (e.g. a ceiling). This allows route planning to retain some of its two dimensional outlook.

Any two enclosures may be placed arbitrarily with respect to each other, except that one enclosure may not overlap or be contained within another. Typically, enclosures within a building will be grouped by stories. A stairway is an enclosure which connects one story to another. Some examples of enclosures would be the rooms in a building, the hollow of a cave, or a tunnel. The enclosure data structure also contains a floor outline of the enclosure and the average height of the enclosure.

Enclosures are made up of "walls", which are triangulated surfaces in which mobility and line-of-sight are both blocked. Each enclosure has a list of wall triangles and a list of apertures, with each aperture individually triangulated. The union of all those triangles is required to completely bound the enclosure, so that there is no path from the interior of

the enclosure to its exterior which does not cross at least one (wall or aperture) triangle. The only exception to this is the exterior enclosure of the MES structure.

2.3 Apertures

Two adjacent enclosures may be connected by one or more "apertures". An aperture is a planar polygonal object of uniform, non-zero transparency, which connects exactly two enclosures. The degree of transparency/visibility is stored in the aperture data structure. Each aperture is triangulated, and any aperture can permit or deny mobility, as determined by the modeller.

Consider the example of a single pane window. Since an MES aperture is required to be planar, the MES aperture corresponding to the window consists simply of the glass portion of the window. The window's casing and sill are triangulated with one or both of the adjacent enclosures. Similarly, a bay window would be represented as several distinct apertures, one for each flat glass area of the bay window.

Apertures have two distinct functions. They describe some of the possible mobility connections between enclosures (i.e., they are the edges of the mobility topology). Also, they group together transparent or semi-transparent triangles of similar transparency. This saves some storage space, since transparency/visibility is stored once for each aperture, not once for each triangle of the entire MES.

An aperture cannot have internal structure. It must have the same transparency over its entire surface. If an aperture has sub-areas with different transparencies, it may be broken down into smaller apertures to maintain uniformity of transparency. However, the topology will not recognize the size of the overall aperture in that case.

An aperture forms an interface between two distinct adjacent enclosures, where one of the enclosures may be the exterior enclosure of the MES structure. For each of the two enclosures adjacent to a given aperture, the aperture has a field specifying whether there is a non-zero "step" from the lower edge of the aperture to the floor of the enclosure, as shown in Figure 1. It is used at run-time as an optimization in determining whether a unit can traverse the aperture. The "step", or discontinuity, need not be immediately

adjacent to the aperture. For example, a window with a flat window-sill, ten feet above the floor, would probably be deemed to have a step discontinuity.

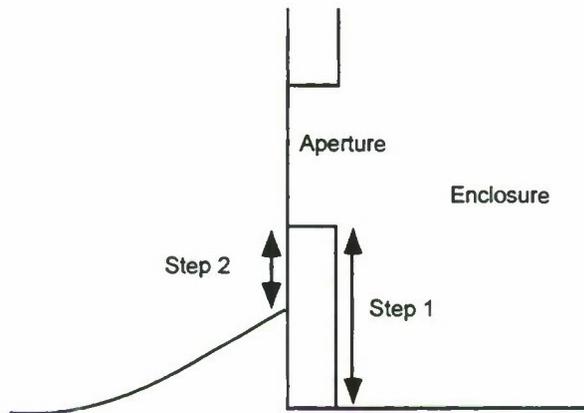


Figure 1: Step Discontinuity at Apertures

In conjunction with the step discontinuity flag, each aperture stores the thickness of the surrounding wall, to allow the run-time code to calculate the height of the step. In the example above, the thickness would be the width of the window sill.

For higher resolution apertures, such as “mouse holes” that result from a detonation, the cost of triangulation may be prohibitive. For these cases, the capability to associate a 32 by 32 bit array with an aperture has been provided, where the values in the array specify the location of the actual aperture. The aperture is still triangulated, but only two triangles, which define the area of extent of the bit array, are used. The algorithms that operate on apertures check these triangles first, and then check the bit array if the aperture is of interest.

2.4 Triangles

The data structure of every triangle (wall and aperture triangles) in an MES structure contains three vertices, specified in the MES’s local coordinate system. The vertices are ordered counterclockwise when viewed from the triangle’s outward face. Triangles also possess a normal vector, so that the direction that the triangle is facing (such as the “top” of the floor) is easily recognizable. Also, the angle between the triangle’s normal and the gravity vector can be used to determine if an entity can reside at that triangle, or if

the angle is too steep. The triangle data structure also contains a material type, which can be one of the Material Composition Category FACC attribute values.

Figure 2 shows the relationships between the various MES data types. MES structures are stored as templates, which contain the apertures, enclosures, and triangles that define the structure. These templates are used to create instances of the structures at specific locations in the database. These instances are stored in the volume data structure. When an instance of an MES structure is changed during run time, a new template is created for the changed structure, which is then referenced by the volume data structure for the changed MES structure.

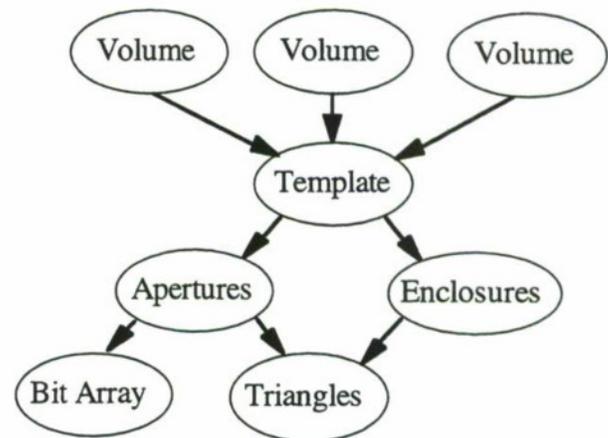


Figure 2: MES Data Hierarchy

2.5 Topology

The relationship between enclosures and apertures, where apertures are the only gateways between any two enclosures, provides sufficient information to generate a connectivity graph for any MES structure. In such a graph, the nodes of the graph represent the enclosures of the MES structure and the edges represent the apertures. This connectivity graph can be used for route planning inside or through MES structures. For example, to plan a route between two points in an MES building, the enclosures that those two points lie within are found, and then a route is generated utilizing the network of enclosures and apertures. A lower level path planner could then plan the actual path inside an enclosure between the apertures of the higher level route.

The topology defined by this connectivity graph yields all possible routes within an MES. However, some of those connections may not be feasible for a given entity. For example, an individual combatant should not be able to enter an MES building through an aperture that is 20 meters off the ground, unless special equipment is used. Therefore, MES structures were designed to provide a feasible topology. Candidate paths can be filtered to disallow movement through closed/blocked apertures, to prevent routing through apertures that require rises in elevation greater than a climb threshold, or to prevent routing through apertures that require a decline in elevation greater than a fall threshold. This filtering is based on information recorded with each aperture: the step discontinuity flag, and the wall thickness flag. If the step discontinuity flag is set for an aperture, the topology traversal routines will check the height on each side of the aperture. This is done by looking up the elevation at a point "wall thickness" away from the aperture, and subtracting the elevation at that point from the elevation of the aperture. This difference is compared to the maximum climb or fall height for the specified entity, as appropriate for entering or leaving the aperture. If the elevation difference is within the limitations of the entity, then that aperture is considered a topological edge. Otherwise it is ignored for the purposes of the specified entity.

3. ModSAF Modifications

The changes made to ModSAF to support MES structures were mostly made within the Compact Terrain Database library (libCTDB). We also modified other portions of ModSAF in order to demonstrate the use of MES structures in route selection and planning.

3.1 Compact Terrain Database

Many changes were made to the Compact Terrain Database (libCTDB) data structures and algorithms in order to support MES structures. These changes include implementation of the various MES data structures discussed in the previous section, modifications of the libCTDB algorithms to work with MES structures, and the addition of new functions to the API for use of the MES structures by the ModSAF application.

3.1.1 MES Data Structures

Besides the enclosure and aperture data structures previously described, other data structures were added to libCTDB in order to support MES structures. Most of these are internal to libCTDB, but one data structure, the MESKEY, is used in the libCTDB API for MES structures. The MESKEY is an aggregate data structure that contains the unique ID of an MES structure, as well as an identifier for a unique enclosure within that MES. The MESKEY is used in route planning to identify those route points that are within an MES structure, and exactly where those points are within the structure.

3.1.2 MES Algorithms

The approach taken for dealing with MES structures within libCTDB algorithms was to modify the existing algorithms, such as intervisibility, elevation look-up and ground intersection, to identify when an MES structure was intersected, and branch to routines that were written specifically for MES structures while within them. For example, when a line of sight calculation is performed from outside an MES building into it, the existing intervisibility routine is used until the line of sight ray intersects the bounding volume of the building. At that point, the MES intervisibility algorithm is used until either the ray is blocked by something within the building or it leaves the building. If the ray leaves the building without being blocked, the processing continues using the non-MES intervisibility algorithm.

The line intersection algorithm is the basis for several of the MES functions used in the intervisibility, elevation look-up, and ground intersection routines. This algorithm calculates the intersection points of a line segment with the triangles which define the MES shape and orders the intersections along the line segment. The algorithm can be limited to finding the first intersection point along the line segment.

In order to calculate these intersections points, the algorithm, in theory, would have to test the intersection of the line segment with each MES triangle. The use of grid boxes has been introduced to significantly reduce the number of these intersection tests. The grid boxes are a hierarchy of boxes, each of which contains a subset of the MES triangles. Each box either contains two or more grid boxes or is a lowest level grid box. The line intersection algorithm determines the sequence of lowest level grid boxes intercepted by the line segment and then, in this sequence, tests each of the triangles contained within

these lowest level grid boxes. Since triangles often are located in more than one lowest level grid box, the algorithm keeps track of which triangles have been tested so that no triangle is tested more than once.

All of the grid boxes are contained within the outer most grid box, which spans the entire MES structure. As discussed elsewhere, MES structures use an internal coordinate system in which coordinates are allowed to vary between -2^{30} and 2^{30} ; the x, y, and z dimensions of the outermost grid box are chosen to be the smallest powers of 2 which allow it to contain the entire structure (the reason for this will become apparent in the following discussion).

When the MES structures are added to the database, the corners of the outer most box are determined, and an iterative process is used to subdivide the boxes. The process is driven by the number of triangles overlapping each box. If this number is more than a specified constant (nominally 12) and the box's longest side is greater than one, the box is subdivided by splitting each of the long sides by two. If the box has two equal length sides that are longer than the third, the box is split into two equally sized boxes along the longest side. If the longest side of the box is longer than one of the other sides, the box is split into four equally sized boxes along the longest two sides. If all sides of the box are equal, the box is split into eight equally sized boxes. Since the side dimensions of the outer most box are powers of two, the splitting process eventually leads to cubic boxes.

Testing whether a triangle is in a grid box begins with testing its three vertices. If any of these vertices is in the grid box, the triangle is in the grid box. If all of the vertices are outside the grid box, the intersection of each side of the box with the triangle is tested. If any side intersects the triangle, the triangle is in the grid box.

The line intersection algorithm calculates the intersection points of a line segment with the triangles which define the MES shape and orders the intersections along the line segment. The line segment is defined by the start and end points, P_0 and P_1 , either of which can be inside or outside of the outer most grid box. A point P on the line segment is defined in terms of the parameter α .

$$P = P_0 + \alpha (P_1 - P_0), \text{ where } 0 \leq \alpha \leq 1$$

The line intersection algorithm first determines whether P_0 is inside or outside the outer most grid box. If the point is inside this box, the algorithm finds which lowest level grid box (one which is not subdivided) contains this point. If the point is outside the outer most grid box, the algorithm determines if the line segment intersects it. If it does, the point of intersection closest to point P_0 is determined and the lowest level grid box at that point is found. If the segment does not intersect the outer most grid box, the line segment does not intersect the MES.

Finding the intersection of the line segment with the outer most grid box is accomplished by testing the intersection of the line segment with all six sides of the box and then calculating which of those intersections is closest to point P_0 .

Finding the lowest level grid box starting with a point in a subdivided grid box is an iterative process as the algorithm first calculates which of the subdivided boxes contains the point. If this box is further subdivided, this calculation is repeated. Otherwise, the lowest level box is found.

Once a lowest level grid box is found, the intersection of the line segment with all triangles that intersect that grid box is determined. If the line segment intersects a triangle, the normalized distance to the intersection, α_i , is used to order the intersections.

A record of each triangle tested for intersection is maintained so no triangle is tested twice. Having tested all of the triangles that intersect a lowest level box, the next lowest level box through which the line segment passes is found. First the intersection of the line segment with all sides of the current lowest level box (except the entry side) is calculated. The distances to these intersections are calculated for the (infinite) surfaces of the sides, not taking into account the extent of the sides. Whichever distance is smallest determines the intersected side. If the normalized distance to this side (α) is greater than one, point P_1 is in the current lowest level grid box and the algorithm is complete.

The algorithm then examines the "parent" grid box containing the current lowest level box to determine if the side through which the line segment is passing is also the side of this box. If this side is not part of the "parent" grid box, the next grid box in the sequence for the "parent" grid box is chosen. If this side is part of the "parent" grid box, the process is

repeated for the “parent” of the “parent” grid box. When the “parent” grid box is the outer most grid box, the algorithm is complete. Once the next grid box is found, if it is not a lowest level grid box the process of finding the lowest level grid box (described above) is repeated. Having found the lowest level grid box, testing the triangles it contains is also repeated. This process continues until either point P_2 or the side of the outer most grid box is reached.

The calculation which determines if the line segment intercepts a triangle uses the two points that define the line segment, the triangle unit normal, \underline{n} and the vertices of the triangle, V_0 , V_1 , and V_2 (Figure. 3). First, the normalized distance, d , to the plane of the triangle along the line segment is calculated.

$$d = \underline{n} \cdot (P_0 - V_0) / [\underline{n} \cdot (P_1 - P_0)]$$

Then, the intercept point of the line segment is calculated.

$$P_i = d (P_1 - P_0) + P_0$$

Note that normalized distance, d , must be between 0 and 1 for the line segment to intercept the triangle. Once the distance is calculated, a vector in the plane of the triangle, pointing into the triangle from the side connecting vertices i and j (and normal to it) is computed for each side.

$$\underline{n}_{ij} = [\underline{n} \times (V_j - V_i)]$$

Note that one normal can be computed more efficiently from the other two.

$$\underline{n}_{20} = \underline{n}_{01} + \underline{n}_{12}$$

If the intercept point P_i is inside the triangle, the dot product of the vector \underline{n}_{ij} and the vector from vertex i to the intercept point must be positive. That is:

$$\underline{n}_{ij} \cdot (P_i - V_i) \geq 0$$

The algorithm tests this criteria for all three sides, i.e. $(i,j) = (0,1), (1,2)$ and $(2,0)$.

The order that the triangles are intercepted along the line segment is based on the value of d .

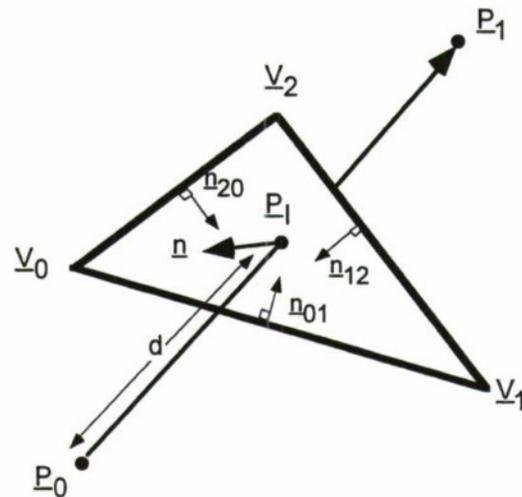


Figure 3: Triangle Intersection

The intervisibility and the ground intersection functions directly use the line intersection algorithm with the wall triangles and those aperture triangles with non-zero transparency. The intervisibility function directs the line intersection algorithm to calculate a specified number of intercepts. The ground intersection function directs the line intersection algorithm to calculate one intercept.

The multiple elevation lookup function uses the line intersection algorithm with only the wall triangles. The line segment is defined using the specified x,y point and a point above the outermost grid box and a point below the outermost grid box. The direction of the normal relative to the vertical line segment does not affect the triangles being tested. The number of intercepts per enclosure is limited to two by the layout of the MES.

3.1.3 MES API Routines

A number of new routines have been added to the libCTDB API. These routines allow qualified elevation lookup for multiple elevations (Stanzione, et. al. 1996), as well as access to the MES geometry and topology information. Table 1 lists these new functions.

3.2 Route Selection and Planning

The modifications to ModSAF to demonstrate route selection and planning within MES structures were based on a depth first thread that allows placement of an individual combatant on the terrain or inside of an MES structure and tasking of that entity to move

into, within, and out of that structure. The challenge in this approach was to implement three dimensional (X,Y,Z) motion in an essentially two dimensional movement environment.

MES enclosure and aperture data. This object is used by the LineClass PO object, which is used to store routes. This change allows route information within MES structures to be stored in the persistent object database.

The PointDescription object in the persistent object (PO) database library (libpo) was modified to store

Table 1: MES API Functions

ctdb_lookup_elevation_mes	Elevation lookup that takes MES structures into account
ctdb_lookup_qual_elevation	Generic elevation lookup routine
ctdb_elev_data_to_string	Converts elevation data into ASCII text
ctdb_sort_elev_data	Sorts a variable length list of elevation data
ctdb_mes_get_topology	Returns the topology of the outside node
ctdb_mes_get_connected_edges	Returns a list of edges that connect to a node
ctdb_mes_filtered_connected_edges	Filtered version of ctdb_mes_get_connected_edges
ctdb_mes_connected_nodes	Returns the two nodes that are connected by an edge
ctdb_mes_get_filtered_connected_nodes	Filtered version of ctdb_mes_get_connected_nodes
ctdb_mes_find_mes	Finds the MES that is closest to a point within a radius
ctdb_mes_xy_in_enclosure	Determines if the point (X,Y) is contained in the enclosure
ctdb_mes_find_enclosure	Finds the enclosure id or node for a given (X,Y,Z) and
ctdb_mes_find_node	MES id
ctdb_mes_get_enclosure	Returns the outline of the enclosure and average height
ctdb_mes_get_node_info	
ctdb_mes_get_edge_info	Returns the outline of the aperture (edge)
ctdb_mes_get_aperture_centroid	Returns the centroid of an aperture
ctdb_mes_get_mes	Returns the roofline vertices that define the MES volume

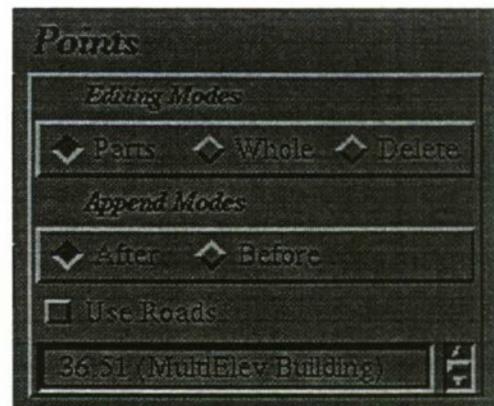


Figure 4: Multiple Elevation Widgets

In order to allow placement of an entity anywhere on the terrain, the user interface was modified to allow a user to select the elevation surface onto which an entity is to be placed. This was achieved by modifying the PLACE and LINE widgets to understand and accept three dimensional inputs, as shown in Figure 4.

When an elevation lookup is initiated from the ModSAF user interface, elevation information for all terrain and volume features at the specified (X,Y) location is determined. All intersections with elevation surfaces are then presented to the user. The user then has the option to select a specific elevation. If the user chooses not to select an elevation surface, the default behavior is to use the elevation of the terrain surface. For example, if the user is specifying a LineClass point, such as a route waypoint, and the terrain elevation is selected, the two dimensional point information (x, y) will be encoded in the SP_Location variant of the PointDescription. If the user selects an MES elevation surface, however, the MES identifier, enclosure identifier, and the two dimensional point information (x, y) will be encoded in the SP_Enclosure variant. In both cases the elevation information can be determined explicitly, since the SP_Location variant is assumed to use the terrain surface, and the SP_Enclosure variant specifies an enclosure, which is defined to have a single elevation at every location.

The route generation utilities were modified to support routes within MES structures. The LineClass object is expanded into a route list, as before, and then is post processed for MES structures. If the route contains any waypoints that are inside of MES structures, the route is modified to allow the entity to move within the MES structure. The enclosure information is stored for each waypoint that is inside an MES structure. This enclosure information, along with the MES topology, is used to generate specific route segments within the MES to allow the entity to move from the previous waypoint, which may be outside or inside of the MES structure, to the waypoint that is inside the MES structure. Similarly, if a waypoint is outside of an MES structure, but the previous waypoint was inside the structure, a route segment is generated to move out of the building from the previous waypoint. As an example, Figure 5 shows a route with one of the waypoints within an MES structure. The final route generated by the route planner is shown, with intermediate waypoints within each enclosure that must be traversed to get to the selected waypoint.

The vehicle move task uses the waypoints in this expanded route as goals which are provided to the lower level path planner (libmovemap). The movement goals and the obstacles in the movemap, however, must be unambiguous in elevation, since the movemap library only plans in (X,Y,T(time)). Since each enclosure is defined to have only one positive (normal up) and one negative (normal down) elevation surface, (x,y) locations within an enclosure are unambiguous, and libmovemap can be used as long as the movement planning is done within each enclosure separately.

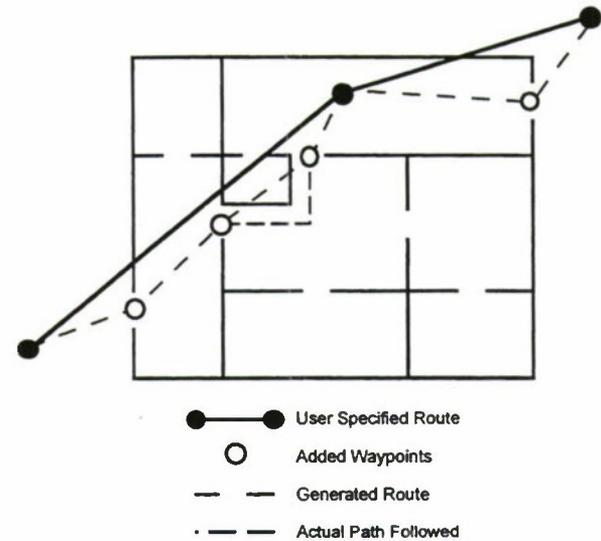


Figure 5: Route Planning in MES Structures

Obstacles generated for placement into the movemap obstacle map are generated by the libvterrain library. This library is being modified to support on demand map generation and MES enclosure obstacle generation. Since the time required to travel between enclosures is relatively short (when compared to the outside terrain movement) rapid, on demand movemap planning is required. It is expected that the time to generate an obstacle map for a single enclosure will be short since the region of interest is small.

The libvterrain library is also being modified to support MES obstacle insertion into a vehicle short term movement map. If the movement is outside of an MES structure, obstacle generation remains unchanged. When planning for movement inside of an MES structure, the obstacle generation code will insert MES related impediments to movement (e.g. walls) into the vehicle's movemap as obstacles. The obstacles inserted into the movemap are the set of

obstacles that are interesting in the enclosure to be traversed. This processing will occur as soon as each new enclosure is entered. In addition to adding static obstacles to the movemap, moving obstacles will also be added if they are contained within the enclosure being traversed. Figure 5 shows the actual path taken by an entity within an MES structure, based on the previously generated route and MES structure as input to movemap.

3.3 Other ModSAF Modifications

The libCTDB terrain viewer and libxcig tools were also modified to work with the MES structures. These modifications allow the developer to test the MES functionality and visualize the MES structures in both two and three dimensions.

The viewer was modified to allow rendering of MES volumes in the plan view and the three dimensional view. In the plan view, intervisibility and elevation lookup options were both modified to work with the MES structures. The intervisibility option allows the intervis ray to extend inside of an MES structure. The elevation lookup option provides a list of the elevations of all of the surfaces at the selected (x,y) location.

The ability to visualize MES structures was integrated into libxcig. In addition to the extraction and display of the polygons of an MES structure, the expanded material identifiers were used to provide easier visualization of the different surfaces of the MES structure. For example, the walls of the MES and the floor are displayed in different colors.

The XCIG demonstration software was modified to allow movement through MES structures. Movement through physical obstacles in an MES, such as the walls, is prevented. In addition, the command line options of the demonstration software were enhanced to allow the specification of a multiple elevation starting location.

4. Compiler Modifications

Support for compiling MES structures into CTDB terrain databases has been added to the **recompile** and **s1kctdb** compiler programs. These changes will be present in CTDB format 6 databases.

In order to get MES structures into CTDB databases quickly, the recompiler correction file mechanism was expanded to support the definition of MES structures outside of the S1000 toolkit. Two new operators are now supported within correction files. **ADD_MES_TEMPLATE** allows an MES structure to be defined, and **ADD_MES_INSTANCE** allows the instantiation of the structure in the database. When one of those two operators is encountered in a correction file, the compiler creates a data structure for an MES template or volume as appropriate, and passes it to the compiler back end for processing. In the correction file, each MES volume is linked to its template by a unique ID, chosen by the modeller.

The following format is used in the correction file to add a new MES template:

```
(ADD_MES_TEMPLATE <template-name>
  (type <type>)
  (meters_per_unit <real>)
  (roofline <vertex> <vertex> <vertex>...)
  (floorline <vertex> <vertex> <vertex>...)
  (enclosures <enclosure> <enclosure>...)
  (apertures <aperture> <aperture>...)
)
```

The following format is used to add a new MES volume, such as a building, a bridge, or a tunnel. **<template-name>** is the link between the new volume and its corresponding template:

```
(ADD_MES_VOLUME <template-name>
  (origin <real x> <real y> <real z>
    <integer cell-id>) ;; GCS
  (mes_to_world_rotation_matrix
    <real> <real> <real>
    <real> <real> <real>
    <real> <real> <real>)
)
```

where **<enclosure>** is:

```
(<enclosure-name>
  (average_height <real>) ;; meters
  (footprint <vertex> <vertex> <vertex>...)
  (apertures <aperture-name> <aperture-name>...)
  (triangles <triangle> <triangle> <triangle>...)
)
```

and <aperture> is:

```
(<aperture-name>
  (visibility <real>) ;; 0 <= visibility <= 1
  (mobility <real>)   ;; 0 <= mobility <= 1
  (connects_to <connection> <connection>)
                        ;; exactly two connections
  (outline <vertex> <vertex> <vertex>...)
  (triangles <triangle> <triangle>...)
  ).
```

<triangle> is defined as:

```
((material <integer>) ; FACC MCC
 (vertices <vertex> <vertex> <vertex>)
   ; exactly three vertices
 )
```

with <vertex>:

```
(<real x> <real y> <real z>
   ; local MES coordinates
 )
```

and <type> is one of:

```
unknown
building
cave
tunnel
bridge
other.
```

<template-name>, <enclosure-name>, and <aperture-name> are arbitrary, non-quoted strings.

We are currently working with the S1000 developers to add support for MES structure information within S1000, which would allow the MES data structures to be populated directly from S1000. This would include not only three dimensional models (buildings), but also multiple elevation terrain features, such as caves and tunnels. We have developed preliminary algorithms for extracting MES information for simple models, such as bridges.

5. Conclusion

The multiple elevation surface structures described in this paper are planned for integration into ModSAF in July 1996. The MES representation provides capabilities to ModSAF that will allow more realistic

vehicle and individual combatant behaviors to be developed that utilize multiple elevations. Particularly, the use of building interiors will now be possible.

6. Acknowledgment

This work is being done as part of contract DACA76-94-C-0022 from the Advanced Research Projects Agency (ARPA) and the US Army Topographic Engineering Center (TEC). The authors wish to thank George Lukes of ARPA and Kevin Mullane of TEC for their interest, encouragement, and guidance.

7. References

- Evans, A., Stanzione, T. (1995), "Coordinate Representations for CGF Systems", 13th Workshop on Standards for the Interoperability of Distributed Simulations.
- Stanzione, T., Chamberlain, F., Evans, A., Buettner, C. (1996), "Ocean Representation in the Improved Computer Generated Forces Terrain Database", 6th Computer Generated Forces and Behavioral Representation Conference.

8. Authors' Biographies

Thomas Stanzione is the manager of the Computer Generated Forces Section at TASC. He is the Program Manager for the ICTDB project and a key contributor to TASC's other CGF programs, including the DIS Exercise Construction Toolset (DISECT) being developed for STRICOM. His interests include data representations for terrain reasoning and terrain database generation for simulation applications. Mr. Stanzione has a Masters of Science degree in Photographic Science from the Rochester Institute of Technology.

Alan B. Evans Dr. Alan Evans is the manager of the Burlington office of SAIC and is the technical lead on the ICTDB project. He has worked in Advanced Distributed Simulation for over five years with primary research interests in simulation systems architecture, object modeling, distributed object technology and synthetic environments. Dr. Evans has a Ph.D. in Mathematics from Michigan State University and an M.S. in Computer Science from New York University.

Cedric Buettner is a Senior Software Engineer at SAIC, Burlington. He has been involved in the

integrated TIN representation, Global Coordinate System, Ocean Representation and the Multiple Elevation Surfaces development under the ICTDB program. Prior to joining SAIC, he worked on Raytheon's Patriot Fire Unit software developing prototype tactical system enhancements. He is a graduate of Gordon College in Wenham, MA with a BS in Physics and Mathematics.

Forrest Chamberlain is a Member of the Technical Staff in the Computer Generated Forces section at TASC. Forrest has been involved in Computer Generated Forces work since joining TASC in 1994. He is currently responsible for the terrain reasoning and mission tracking components of the Command Forces (CFOR) project at TASC and is a critical contributor to the ICTDB terrain representation effort. Prior to joining TASC, Forrest participated in the hardware and software design of a "wearable" computer system at Carnegie Mellon University, where he earned his Masters Degree in Electrical and Computer Engineering. Forrest earned his BS in Electrical Engineering at Cornell University.

Jonathan Fisher has been a Software Engineer at SAIC since September of 1995, working on Integrated CGF Terrain Database development. Prior to joining SAIC, he was employed by a Seattle area company providing software support for AIDS research labs through the NIH. Jonathan graduated from Dartmouth College with a BA in Math, Physics, and Computer Science, and an MA in Math.

Howard Lu is a Software Engineer at SAIC. Since joining in August 1995, Howard has been involved in the Integrated Computer Generated Forces Terrain Database development and the Synthetic Environment Data Representation Interchange Specification program. He graduated from the Massachusetts Institute of Technology in Cambridge, MA with a MS in Computer Science in 1995.

Lawrence E. Mabi is a Principal Member of the Technical Staff in the Computer Generated Forces section at TASC. He is currently involved with the development of the Multiple Elevation Structure component of the ICTDB terrain database. He is also the principal developer of a reconfigurable simulation and analysis tool for Rome Laboratory. Lawrence received his BS and MS in Mechanical Engineering and PhD in Computer and Systems Engineering at Rensselaer Polytechnic Institute.

Mike Sousa is a Coop Engineer for the Systems Development Section at TASC. During his coop, Mike has been involved with the Joint Precision Strike Demonstration (JPSD) project since starting at TASC in 1994, and more recently with the ICTDB terrain representation effort. Mike is currently working towards the completion of his Masters Degree in Systems Engineering at Boston University.



Representations of Buildings for Individual Combatant CGF

Douglas A. Reece and Hsiao-Kun Tu
Institute for Simulation and Training
3280 Progress Dr., Orlando, FL 32826
drece@ist.ucf.edu, hktu@ist.ucf.edu

1. Abstract

Two areas of increasing interest to the military training community are individual-level simulators and urban environments. For the purposes of trainee platforms, buildings may be represented as sets of polygons in a format suitable for commercially available image display software and hardware. This representation is inadequate for Computer Generated Forces (CGF). A CGF soldier simulator must check visibility, determine height, and detect collisions—all potentially expensive computations. In addition, CGF soldiers must plan and control movement, perform basic spatial reasoning, and make tactical plans. The raw polygonal representation is unsuitable for these tasks.

In this paper we describe our efforts to take a raw polygonal building description and automatically process it to produce new representations. The new representations provide an efficient organization of polygons for visibility and height calculations, simplified obstacle models for other geometric calculations, and semantic information for reasoning tasks. Such automatic processing tools will be valuable for rapidly generating urban databases for simulation exercises involving CGF.

2. The Need for CGF Building Representations

The Institute for Simulation and Training (IST) is developing autonomous computer controlled hostiles and neutrals (CCH/N) to populate a virtual battlefield as part of the Team Tactical Engagement Simulator (TTES) project. This project, which is sponsored by the Naval Air Warfare Center Training Systems Division in Orlando, will develop a system to train small infantry units to fight in urban terrain. The terrain database used for TTES is in a format designed for image generators with no organization or semantic information suitable for CGF. While this is not a big problem for the ground surface, for buildings—of which there are many in TTES—it is. In this paper we describe our efforts to deal with the building part of the TTES terrain database. First we

discuss the characteristics of the source data and buildings and review some of the needs of CGF with respect to terrain. Next we present the data representation that we have developed so far to support these needs; in particular we discuss the height and intervisibility functions. Finally we describe the algorithms used to extract the CGF building representation from the source data.

2.1 Source Representation

The source data for a TTES building is simply a set of polygons. The polygons are contained in a file in Multigen Flight format. This is a very common format for visual databases as it is supported by common visualization hardware and software (i.e. Performer running on SGI workstations).

In Flight files, polygons can be grouped, and the groups can be arranged hierarchically. While it is possible to build a file whose structure maps well to, for example, the topology of a building's rooms or the structure of its walls, in practice it is common for the groups to correspond to polygons that have something else in common. For example, if all windows of the building look the same, the database designer might construct one window frame, then replicate it around the building and put all window frames in a group. The TTES buildings files have no groupings useful for a CGF database.

The raw polygonal representation lacks semantic information that one would intuitively expect to be useful for operating in a building. People commonly see buildings as collections of rooms and hallways joined together by doorways; windows and doors connect the inside with the outside. Buildings have levels, and stairs (and elevators, etc.) connect them. Buildings can also be viewed as spaces partitioned by structural walls. The raw polygonal database contains none of this information. Of particular note is the lack of aperture information; in the raw data, an aperture is the *lack* of polygons.

2.2 CGF requirements

In order to determine what representation was needed for CGF, we first had to examine the functional requirements of CGF with respect to terrain. This section reviews some of these requirements. The first two, height and intervisibility, are common to many CGF systems; the remainder of the requirements are based on functions in the TTES CCH/N system, but are similar to functions in other CGF systems.

- **Height of terrain.** Since gravity pulls entities down to the surface of the terrain, it is necessary to determine the surface elevation of any (x, y) point. Clearly, buildings can have multiple surfaces at each (x, y) .
- **Intervisibility.** Basic CGF behavior are based on whether or not an entity can see another entity. Thus CGF must be able to determine if a line segment from them to another entity intersects any terrain polygons.
- **Collision detection.** As entities move, they must not penetrate any impenetrable obstacles. In our CCH/N system, entities are approximated by a two-dimensional (2D) bounding curve, and they are checked for intersections with line segments that represent obstacles.
- **Dynamic movement control.** Dynamic movement control has similar requirements to collision detection: as the entity moves, it monitors the relative positions of obstacles (moving and static) and makes corrections to its speed and direction of movement. Obstacles are again represented by line segments.
- **Movement planning.** The CCH/N system includes a mode of movement that involves pre-planning a route in clear space between static obstacles from one point to another. Route planning is done in 2D; this is a good abstraction for ground-based entities because it captures most movement constraints and makes computation much simpler. Planning can be done by searching a movement grid upon which obstacle line segments have been drawn. The grid cells can be marked to indicate a cost based on trafficability, exposure to threat, etc. Apertures in or adjacent to a cell can just be treated as a factor increasing the cost to enter the cell. Our first goal for extracting information for movement planning is thus to provide a list of obstacle and aperture locations (line segments).

Rather than searching a large movement grid, it is more efficient to plan first in an abstract space. In a building such an abstract search is most easily constructed by assuming that apertures separate

and connect clear spaces; the movement planner can then first search a graph of apertures and secondly search the free space between apertures. Our second goal to support movement planning is therefore to build a graph of apertures and clear spaces.

- **Tactical reasoning.** The above requirements address *how* a CGF entity moves. Beyond this, the CGF entity must decide *if* and *where* to move. Most of this reasoning can be performed just by knowing the movement points between points and whether one point is visible from the other. An entity could use this information to answer such questions as “Where can I move to to see point P ? What location is near cover but allows me to fire at P ? Extra information allows heuristics to be used to narrow the search for tactically good positions: the boundaries of apertures, the locations of inside and outside corners, etc.

2.3 TTES Buildings

The TTES database is a representation of the Combat Training Village in Quantico, Virginia (QCTV). It contains about 16 buildings typical of those that might be found in a small town. The largest are three stories tall and about 20 meters on a side.

While the obvious way to represent buildings might be as a box divided into levels and further into rooms, we have found numerous characteristics in the TTES buildings that make simple representations and algorithms problematic. Some of these characteristics are as follows:

- **Apertures.** The TTES buildings have “normal” doors and windows, but also “loopholes” the size of a cinder block (20cm x 40cm), windows with low sills that a person can walk through, high windows above a person’s head, and low windows by a person’s feet. There are also apertures in floors.
- **Porches.** Many exterior doorways exit the buildings onto porches which have stairs to the ground. These porches are open to the outside on one or more sides and are effectively part of the terrain surrounding the building, but their polygons are part of the building definition.
- **Balconies.** Several buildings have balconies that are partly enclosed, like the porches described above. One balcony is cantilevered over the ground. An entity can thus be under a building floor polygon but be standing on the ground surrounding the building.

- **Crawl spaces.** There are crawl spaces under some buildings where entities can move. These are about a meter tall.
- **Varying ground height.** The ground around the buildings varies in height. In many cases the ground is below the level of the first floor; hence the need for porches with stairs. The distance below a ground floor window is therefore different on either side. There are buildings for which it is possible to enter the ground floor from some apertures and the basement from others.
- **Concentric rooms.** Figure 1 shows a building, the "Hotel," in the QCTV with a hallway that follows the perimeter of a rectangle. Inside this rectangle is a stairway that follows three sides of a rectangle. Inside the stairway there is an

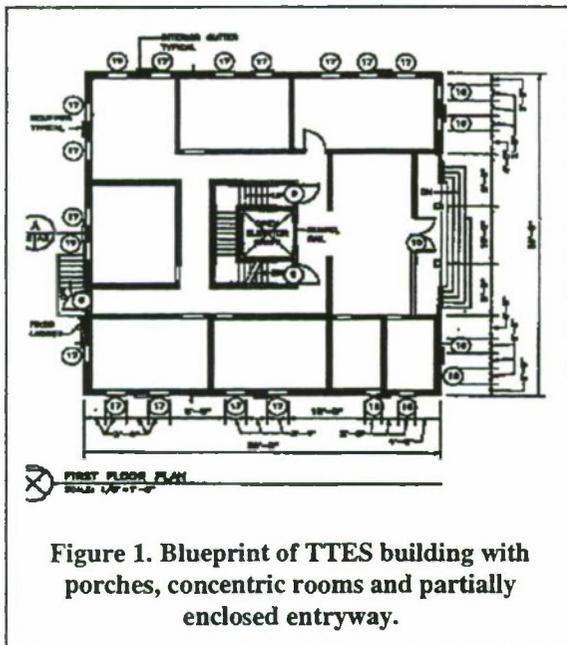


Figure 1. Blueprint of TTES building with porches, concentric rooms and partially enclosed entryway.

elevator shaft. This building also has two porches and an entryway that is open to the outside on one side.

- **Rubble.** Several buildings have been constructed to appear partially destroyed. These buildings are missing large sections of walls, ceilings, or whole levels. The remaining partial walls have uneven tops; some are traversable, some are effectively full walls, and some vary continuously between the two extremes. Several apertures have very ragged edges.

The one characteristic of the buildings that makes terrain algorithms easier is that they are made up of mostly orthogonal polygons. In the local coordinates of the building, the polygons are normal to the

coordinate axes. This characteristic allows us to use 2D computations in many parts of the algorithms.

3. Representation

3.1 Symbolic Information

The goal of building analysis is to construct a building database that uses data representations suitable for efficient computation of the information described in 2.2. We have chosen the following representations to meet the needs of CGF:

- Two dimensional representations of *Movement Spaces*.
- Multiple 2D Movement Spaces. At the very least, Movement Space must provide an unambiguous height for every (x, y) location, so multi-story buildings require multiple Movement Spaces that generally correspond to the building floors. We will outline how we are extracting room representations so that in the future we may use a separate Movement Space for each room.
- Outdoor Movement Space. Since in general the ground outside a building does not correspond to a Movement Space inside the building, a separate Movement Space is required for entities moving outside the building.
- Polygon floor maps. The 2D floor area of a Movement Space is tessellated into rectangular polygons which are attached to the Movement Space. These polygons, along with a range of elevation values for the Movement Space, are used to organize the polygon search when mapping an (x, y, z) location to a Movement Space.
- Obstacle list. Obstacles to movement in a Movement Space are encoded as a list of line segments and stored with the Movement Space. These are used both for movement planning and for collision detection.
- Aperture list. Apertures contain fields for class, geometry, and connected Movement Spaces. Class indicates the type of aperture, such as transition point, doorway, window, loophole, etc. Transition points are boundaries between Movement Spaces, regardless of whether there is a spatial constriction also present. Geometry information contains the bounding box, and relative height to the surrounding ground or floors. Apertures are linked to the Movement Spaces they connect. Since we currently use entire levels for Movement Spaces, Movement

Spaces may include apertures that do not connect to other Movement Spaces.

- Polygons. As we describe below, all original polygons describing the building are kept in our building representation. Eventually we intend to keep a list of wall, floor, and ceiling polygons with each Movement Space. However, we have not yet implemented this step.

3.2 Polygon Organization

All polygons in the source file are also kept as part of the building representation. As we describe below, we take advantage of the fact that most of the polygons are oriented normal to coordinate axes in the local coordinate system of the building. To make these algorithms easier, polygons are stored in four groups according to their orientations—xy, xz, yz, and “other.” Furthermore, the polygons in each group are sorted along their common normal direction.

4. Terrain Attributes from Location

4.1 Terrain Height

One of the most common requests from the terrain database is the height of the terrain at a location (x, y) . This is generally a unique value on the ground; in a building, however, there may be many support surfaces at a given (x, y) . Our system defines a function `Entity_Height` which determines the height of the terrain for an entity. This function is the same as the basic `Height` function when the entity is over the ground, but uses the multiple height values and the old elevation to determine the new height in a building.

When (x, y) is inside a building, the `Height` function basically searches through the building polygons to see which ones intersect a vertical line that passes through (x, y) . This procedure is essentially the same as searching through the polygons of a ground patch to find the one under (x, y) . If it is inside, then the z of the polygon at (x, y) is computed.

Since the polygons in our building representation are organized into orthogonal sets, the `Height` function can ignore all polygons in the xz and yz sets. These polygons are vertical and cannot be support polygons.

The `Entity_Height` function is similar to `Height` but takes an old z value as a parameter. The function

assumes some coherence in space to disambiguate between multiple possible z values at an (x, y) . In addition, the function is able to prune away many polygons by taking advantage of the fact that the xy polygons are sorted in z . The algorithm works as follows:

1. Given (x, y) , z_{Old} , and xy polygon set S ordered in increasing z ;

2. Let

$$z_{Test} = H_{climb} + z_{Old},$$

where H_{climb} is the maximum step up an entity can make during nominally horizontal movement.

3. Let $i = N$, the number of polygons P_i in S ; decrement i until

$$z \text{ value of } P_i < z_{Test}.$$

4. Let $j = i$, determined above; decrement j until

(x, y) is inside of the projection of P_j onto the xy plane.

5. Determine z for (x, y) in P_j .

This algorithm generally works well for the two dimensional movement abstraction commonly used for ground entities; it finds the first support polygon underneath of the entity. The H_{climb} adjustment allows the entity to traverse a stair riser without causing the algorithm to drop the entity down to a polygon below the step. This adjustment can lead to a problem, however, if the minimum height of a ceiling (in a crawl space, for example) is smaller than H_{climb} .

4.2 Movement Space Identity

Since movement in a building is regulated by Movement Spaces, it is necessary to be able to identify the Movement Space containing a point (x, y, z) . Each Movement Space contains a set of rectangular polygons describing the shape of its floor. The algorithm for identifying Movement Spaces is much like the `Entity_Height` function, except that Movement Space polygons are examined. When a candidate Movement Space is identified, the input z is compared to the elevation range of the Movement Space to determine if the point is part of that Movement Space.

The algorithm above is not very efficient in theory. In practice, we can use one floor polygon describing the bounding box of the Movement Space. The only case in which this simplification doesn't work is for non-rectangular Movement Spaces combined with an input z that is not on a support surface. Eventually, as mentioned above, we will link the source building polygons to Movement Spaces and so determine Movement Space membership directly during the height calculation.

5. Visibility

The second major algorithm for CGF is intervisibility, or clear line of sight (LOS) determination. In our system, this algorithm simply returns a Boolean value indicating whether there is a clear LOS. Our LOS algorithm takes advantage of the fact that most of the building polygons are normal to one of the coordinate axes in local building coordinates. Our building data structure stores these polygons separated into different sets and sorted within the set. The algorithms can treat each set of polygons as a separate case and perform calculations in only the appropriate two dimensions. Since the polygons are sorted, the algorithm can quickly prune some irrelevant polygons. Furthermore, the algorithm uses heuristics to order the sets. To do all of this, the algorithm must of course transform the endpoints of the candidate line of sight from world coordinates to local building coordinates. This is done using transformation parameters stored with the building.

5.1 The Line of Sight Algorithm

For lines of sight that begin and end outside of a building, the LOS algorithm first test to see if the line intersects the bounding box of the building. If so, the building polygons are checked. The points where the candidate LOS pierce the bounding box are calculated and then transformed to local coordinates.

The four sets of building polygons are checked in order of decreasing component length of the candidate line of sight. In other words, if the z component of the candidate LOS is the longest, the xy polygons are checked first, and so on. This heuristic is intended to maximize the likelihood of finding an intersecting polygon in the first set. The set of polygons not orthogonal to a coordinate axis are always checked last because the general three dimensional intersection test is the most expensive.

Within each of the orthogonal polygon sets, the algorithm takes advantage of the sorting to skip over polygons that are outside the range of the candidate line of sight. For example, if the candidate LOS goes from point A to point B , and

$$LOS_{Min Z} = \text{Min}(A_z, B_z)$$

then in the xy set the algorithm skips over all polygons P for which

$$P_z < LOS_{Min Z}$$

Similarly, the algorithm always stops examining the xy set when

$$P_z > LOS_{Max Z}$$

For those polygons in the range to be tested, the algorithm first tests to see if the bounding box of the polygon (computed on the fly) overlaps the bounding box determined by A and B . If so, the algorithm computes the point T where \overline{AB} pierces the plane of P . This calculation is fairly simple since the plane of P is orthogonal to a coordinate axis. A further check is made to see if T is in the bounding box of P . If so, then a two-dimensional Inside_Polygon test is made to see if T is inside P . This test checks to see if the sign of the cross product

$$(T - P_i) \times (P_{i+1} - P_i)$$

is the same for all i , where P_i is the i^{th} vertex of P .

The polygons in the last set are not sorted and cannot be pruned as readily. The algorithm uses a three dimensional bounding box check between the LOS and the polygon to see if the polygon must be tested further; a three dimensional line-polygon intersection check is then made. This check again requires that the point T where \overline{AB} pierces the plane of P be computed; in this case it is computed in full three-dimensional generality. However, the test to see if T is in P is performed in *two* dimensions using the projection of P and T on the xy plane. In the infrequent case that P is vertical, then T and P are project onto another coordinate plane. The two dimensional Inside_Polygon test is much cheaper than a three-dimensional version.

5.2 Experiments

We conducted several experiments to determine the value of using various heuristics and checks described

above. In one experiment we generated 10,000 point pairs randomly distributed inside a building and timed how long it took to check the line of sight between each pair. In one case we used a general, three-dimensional line-polygon intersection calculation; in the other case we used the algorithm described above. The first case took 49 seconds (60 MHz Pentium PC), while the second took 7 seconds. This result confirms that our intervisibility algorithm is significantly more efficient than general LOS algorithm.

We considered dividing the building volume up into three-dimensional grid cells, each containing a list of the polygons that intersected it. This should make searches for polygons intersecting a line more efficient. However, the irregular and sparse nature of the polygon distribution in a building would seem to require a more sophisticated structure (such as an octree) to make the search efficient. We believe that our polygon organization allows an efficient pruning of polygons during the intersection search. We would like to compare the algorithms experimentally sometime in the future.

6. Building Analysis

The goal of our building analysis algorithms is to generate the building representation described in section 3.1. In all steps of this analysis we make tests to determine where clear space is and where solid surfaces are. In order to avoid many expensive geometric computations, we first convert the polygonal representation into a three dimensional image representation and analyze this volume. We digitize the volume in steps of 0.1 meters so that any geometric information that the analysis produces will be accurate enough for the simulation.

6.1 Preliminary Steps

6.1.1 Creation of a Volumetric Image

The terrain and building databases are first scanned into a three dimensional (3D) volumetric image. The size of the 3D volumetric image is 1 meter larger than the bounding box of the building so the ground surface can be included in the 3D image. Each volume element, or voxel, in the 3D volumetric image represents a 0.1 meter cube. Each voxel may have one or more of the following attributes:

- Occupancy: SPACE or SOLID
- Ground relation: GROUND_SURFACE, BELOW_GROUND, or ABOVE_GROUND

- TRANSITION_POINT
- APERTURE
- STANDABLE

In the following algorithm description, we will refer to a function $V(i, j, k)$; this is the access function for the type of voxel (i, j, k) .

All voxels are initialized as SPACE type, which indicates free space, prior to scan converting building polygons and ground polygons. A voxel is set to SOLID type if its bounding box intersects any building polygons. The resulting 3D volumetric image is the approximate miniature of the actual building described by the polygonal database. Dimensions of apertures in the volume image are always smaller than the actual dimensions.

After scan converting the building polygons, the ground surface polygons around the building are converted. A voxel is set as GROUND_SURFACE type if its location is at the height of any ground polygons. A voxel is set as BELOW_GROUND type if it is below any ground polygons. A voxel is set to ABOVE_GROUND type if it is above any ground polygons. The GROUND_SURFACE voxels are later used to identify the characteristics of the perimeter of the building.

Figure 2 is one horizontal slice of the 3D image of the Hotel. The dark lines are SOLID voxels indicating walls.

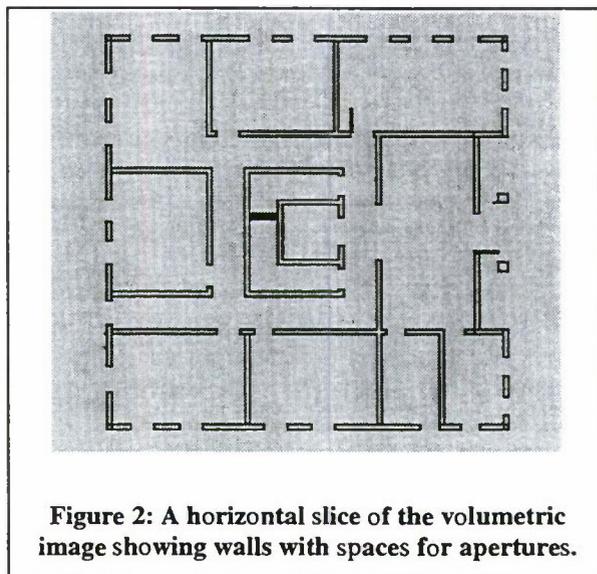


Figure 2: A horizontal slice of the volumetric image showing walls with spaces for apertures.

6.1.2 Filling in Walls

Since the building polygons represent the surfaces of the walls, our algorithm attempts to fill in the core of the walls in the 3D image. This prevents later stages of the algorithm from identifying the space between the walls as a room. The algorithm sets every SPACE voxel to SOLID voxel if it is directly between two SOLID voxels. With a voxel size of 0.1 meters, this procedure will fill in walls that are between 0.1 and 0.2 meters thick. The result of filling process on the Hotel is shown in Figure 3.

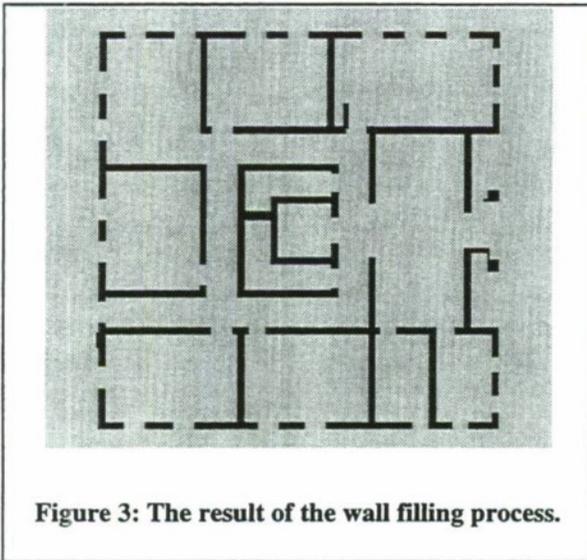


Figure 3: The result of the wall filling process.

6.2 Finding Movement Spaces

6.2.1 Standable Voxel Extraction

The first step in identifying Movement Spaces is to find those voxels on which an entity could stand. These are the STANDABLE voxels. A STANDABLE voxel is defined as a SOLID voxel with a certain number of SPACE voxels directly above it, i.e.

$$\forall \text{ voxels } (i, j, k),$$
$$\text{if } V(i, j, k) = \text{SOLID and}$$
$$V(i, j, k+t) = \text{SPACE}, 1 \leq t \leq N,$$
$$\text{then } V(i, j, k) \Leftarrow \text{STANDABLE.}$$

N is the number of voxels corresponding to $H_{\text{CeilingMin}}$, the minimum height of a ceiling over a STANDABLE voxel. In our implementation, $H_{\text{CeilingMin}}$ is 0.5 meter, so the definition of STANDABLE voxels admits places where crawling is required.

6.2.2 Movement Space Identification

The identification of Movement Spaces has two primary steps. In the first, all standable voxels that are "adjacent" are linked into regions. In the second step, these regions are split so that no region has two standable voxels at the same (i, j) .

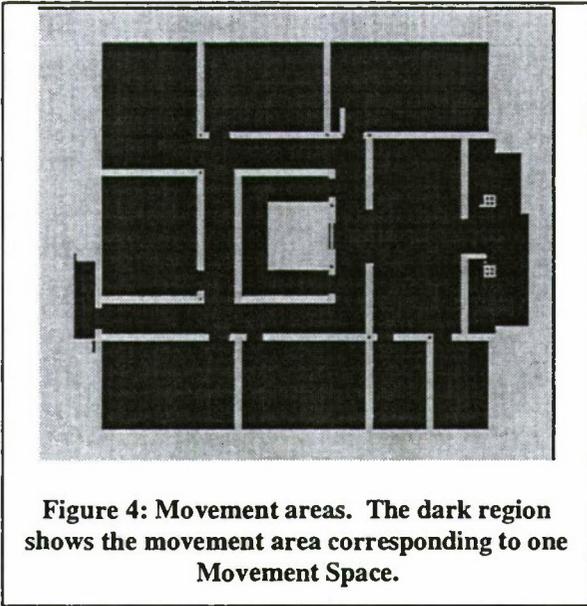
The standable voxel linking step is essentially a three dimensional version of a standard two dimensional region labeling algorithm for image processing. The one notable difference is that standable voxels are considered adjacent if they are adjacent in the x and y directions and within H_{Move} of each other in z . H_{Move} is a threshold height such that an entity can climb this height during horizontal movement with no penalty. This is just an approximation, but at any rate H_{Move} is intended to be much smaller than the H_{climb} used in section 4.1 above.

We recognize that the use of such thresholds makes the analysis of buildings entity dependent. However, this is appropriate; the information we are extracting from the building is intended to be used by individual combatants. Armored vehicles, for example, would view buildings differently so the representation we are extracting would not be very useful anyway.

The second step of Movement Space identification is the division of the standable regions into new regions that do not overlap. Each standable region is considered in turn. It is examined from minimum k to maximum k . At each k , if voxel (i, j, k) in the region is standable, then (i, j) in a two-dimensional region map is annotated with the value of k . This process continues until at some k , for all voxels (i, j, k) in the region, the 2D map is already marked at (i, j) . At this point the 2D map is saved as the movement area for a Movement Space and the voxels corresponding to this area are cleared from the region under consideration. The boundary voxels between Movement Spaces are marked as TRANSITION POINTS. The minimum and maximum k values of the area are found and stored for use in the Movement Space Identity function (described in Section 4.2). The examination of the standable region then continues with a new 2D map. Figure 4 shows one of these 2D maps for the Hotel.

6.2.3 Polygonal Floor Map

We currently generate only a single rectangular polygon to describe the footprint of the Movement Space. This is determined easily by finding the



extreme points of the 2D movement area just computed.

6.3 Characterizing the Outside Movement Space

The Movement Space for the outside is generated much the same as the others, except that ground surface voxels are used instead of STANDABLE building voxels, and the outer boundary of the movement area is not considered. The algorithm must identify obstacles between the building Movement Spaces and the outside, and transition points where an entity can walk freely between them.

Obstacles are formed by walls on the perimeter of the building, where building and ground voxels meet. Voxel (i, j, k) is an obstacle if

1. there is an adjacent voxel (l, m, n) that is of type GROUND_SURFACE, where $k - n > N$, and
2. $V(i, j, h) = \text{SOLID}$ for $n < h < k$

where N is the number of voxels in H_{Move} .

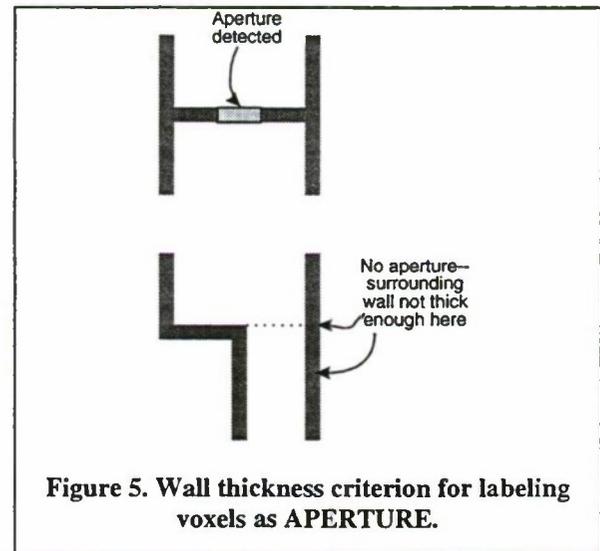
Transition points from the outside to an interior Movement Space may identify doors or windows, but also places where a person can just walk onto a building polygon. Porches are the common example of features that give rise to non-aperture transition points. The definition of transition points for the outside Movement Space is a voxel (i, j, k) that has a GROUND_SURFACE voxel adjacent in (i, j) and within H_{Move} in k .

6.4 Finding Apertures

Apertures are essentially boundaries where Movement Spaces join, but across which entities cannot move without a cost (if at all). They are characterized by a spatial constriction.

6.4.1 Detecting apertures

The aperture extraction algorithm looks for voxels that are in doorways and windows by checking to see if there are SOLID voxels on either side of it (at the same z) within distance $W_{Aperture}$. However, if the wall found on either side is SOLID for less than a thickness $W_{WallMin}$, then the constriction is assumed to be part of a hallway and not an aperture. This condition is illustrated in Figure 5. All voxels meeting the aperture criteria are marked as APERTURE type. The result for the Hotel is shown in Figure 6.



After the APERTURE voxels have been marked, a 3D connected component algorithm is run over the 3D image to connect together voxels into apertures regions. The bounding box is extracted. Since the aperture regions are as thick as the wall they are in, a center point is computed (depthwise). Finally, an aperture list is created and filled with all of the geometric information.

In a similar manner, all TRANSITION_POINT voxels are grouped together. These are two dimensional features so do not carry the same geometric information as apertures. However, they still have the connectivity information.

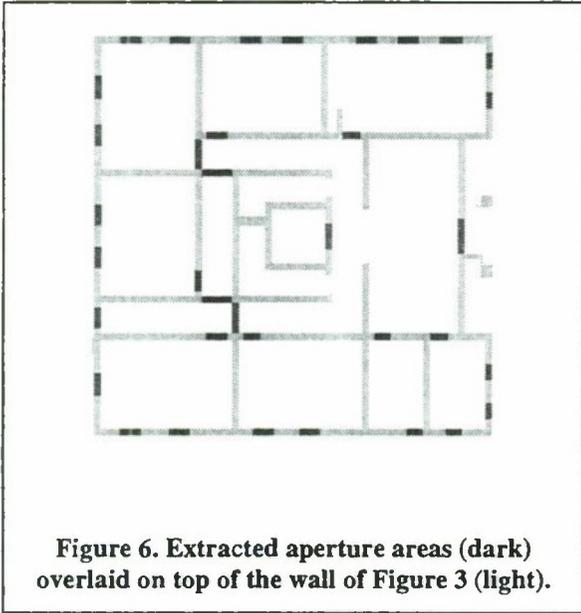


Figure 6. Extracted aperture areas (dark) overlaid on top of the wall of Figure 3 (light).

6.4.2 Connecting Apertures to Movement Spaces

As the apertures placed on a list, they are connected to the Movement Spaces that they adjoin. The aperture has two pointers to its adjacent Movement Spaces and the Movement Spaces have lists of pointers to adjacent apertures. The connection process is straightforward: the (x, y) location just to the sides (depthwise) of the aperture are computed, and the Movement Space Identity function is called at (x, y, z) where z is the elevation of the bottom of the aperture. As Section 4.2 described, the Movement Space corresponding to that point can be obtained by comparing the z with the range of elevations covered by the Movement Spaces.

6.5 Finding Obstacles

The major operational component of Movement Spaces is the list of obstacles. The building analysis algorithm takes the 2D movement map for each Movement Space and identifies all cells that are not in the movement area but are adjacent to a cell that is. These identified cells form the boundaries of the movement area. The cells are obstacles, with the following exception: for each of these cells (i, j) , the algorithm looks at voxel (i, j, k) , where k is the value of cell (i, j) from the map generation process. If this voxel is not solid, then the boundary cell is cleared and the adjacent movement area cells are marked as transition points (this is a place where the floor drops off to another Movement Space). All of the remaining marked boundary cells are clustered into line segments and stored in a list.

Note that the boundary cells include locations that have windows. Thus the extracted obstacles will include areas of the walls with windows. This inclusion is deliberate. The extracted obstacles are used as is to detect the collisions of walking entities, which is the most common case; we desire a collision result at a window if the entity is walking. If the entity is climbing instead of walking, then the collision detection algorithm must consider the wall obstacle only while the entity is below the sill of the window being climbed through. Windows with very low sills, i.e. $< H_{move}$, are already part of the movement area of the movement space and so need no special treatment (they are effectively doorways). For movement planning purposes, the terrain features are written into a grid overlaid on the terrain; the gridding algorithm must simply overwrite wall obstacles with the appropriate aperture and transition point features and the grid will represent the desired mobility characteristics of the wall.

6.6 Future Work

In the future it may be desirable to create smaller movement area objects corresponding to rooms. These room objects intuitively would correspond to topological building features that humans think about when they describe buildings. As such, room objects would be useful for writing terrain reasoning functions for CCHs. The smaller regions represented by rooms could also provide a form of spatial indexing into the building database to allow faster search of obstacles, apertures, etc. during the various terrain functions. We have already experimented with extracting rooms; the main part of the algorithm is connected component analysis. shows a slice of the Hotel after such an analysis. The resulting regions could be treated as Movement Spaces.

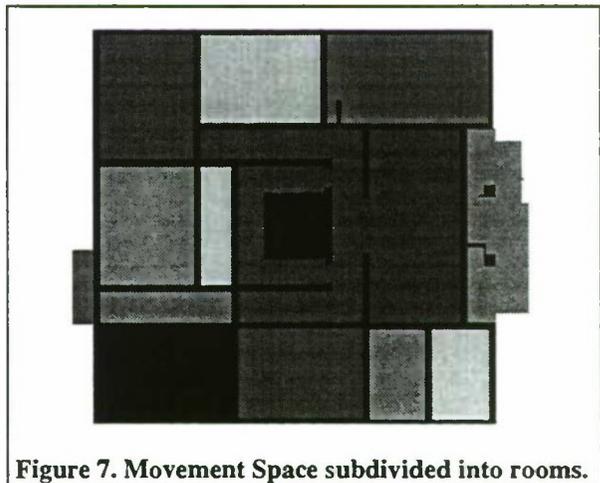


Figure 7. Movement Space subdivided into rooms.

Another useful piece of information to add to the building representation would be the list of polygons associated with each Movement Space. Since the building source files are not constructed with polygons associated with only one floor or room, there could be a many-to-many assignment of polygons to Movement Spaces. Nevertheless, the spatial indexing provided by the Movement Spaces would allow faster access to the building polygons for the functions that require them (e.g. height and intervisibility).

7. Conclusions

We have presented a set of algorithms for taking an unstructured set of polygons describing the surfaces of a building and extracting topological and geometric information useful to a CGF human.

Early in the TTES CCH project we were faced with a choice of building a CGF building database by hand or building tools to do it semi-automatically. We experimented with editing the source files by hand and creating semantic databases from scratch. However, not only would this have been tedious to do once for all buildings, but we received several updated databases during the course of the project. We would have had to re-extract all of the geometric information from each new version. We thus opted for the automatic conversion tools.

We believe that, suitably developed, such tools will be useful beyond TTES for creating representations of buildings for other CGF databases (e.g. Stanzione 1995). Purely polygonal representations for image generators are very common and seem to be the standard format for objects created or recreated on CAD systems. Automatic conversion tools would avoid the time consuming, tedious process of extracting the CGF-specific information from these polygonal representations. Urban databases could thus be constructed more easily and rapidly.

8. Acknowledgment

This work is being supported by contract N61339-94-C-0006 from the Naval Air Warfare Center Training Systems Division.

9. References

Stanzione, T., Chamberlain, F., Evans, A. and Buettner, C. (1995) "Integrated Computer Generated Forces Terrain Database", In *Proceedings of the Fifth Conference on Computer Generated Forces and Behavioral Representation*, University of Central Florida.

10. Authors' Biographies

Douglas A. Reece is a Computer Scientist at the Institute for Simulation and Training. He is the Principal Investigator of the TTES Computer Controlled Hostiles project. His research interests are in artificial intelligence, specifically intelligent agent design and computer vision. He has a Ph.D. in Computer Science from Carnegie Mellon University and B.S. and M.S. degrees in Electrical Engineering from Case Western Reserve University.

Hsiao-Kun Tu is a Computer Scientist at the Institute for Simulation and Training. Currently his effort concentrates on design of a new generation of terrain database for CGF. His research interests are in image processing, pattern recognition, and computer vision. He has received a M.S. and B.S. degrees in Computer Science from University of South Florida.

Ocean Representation in the Improved Computer Generated Forces Terrain Database

Thomas Stanzione
Forrest Chamberlain

TASC
55 Walkers Brook Drive
Reading, MA 01867
tstanzione@tasc.com
flchamberlain@tasc.com

Dr. Alan Evans
Cedric Buettner

SAIC
Suite 130
20 Burlington Mall Road
Burlington, MA 01803
aevans@bos.saic.com
buettner@bos.saic.com

1. Abstract

The Improved Computer Generated Forces Terrain Database (ICTDB) project, being developed jointly by TASC and SAIC, is one of four projects in the ARPA/TEC Advanced Distributed Simulation Synthetic Environments program. The goal of this project is to design and develop the next generation terrain database representation for Computer Generated Forces (CGF) systems. The ICTDB project is focusing on a number of areas of improvement for CGF terrain representations. One of the areas that ICTDB is addressing is the representation of multiple elevation surfaces. These include features that can be overlaid onto the terrain surface, such as water surfaces over the ocean floor, river, and stream beds. Another area is the identification and representation of advanced features and attributes, including all features and attributes normally found in operational terrain sources, such as Interim Terrain Data (ITD) and Tactical Terrain Data (TTD). Features and attributes necessary to support dynamic terrain representations will be provided, as well as a mechanism for easily expanding the feature and attributes represented. ICTDB is providing additional support for dynamic terrain. The ICTDB representation is being designed to be updated in real-time based on information from the DIS network. This capability will allow ICTDB changes based on input from the other ARPA Synthetic Environment programs.

In this paper, we describe the extensions that the ICTDB project has made to ModSAF in each of these areas in order to provide a higher fidelity ocean representation. The sea floor is being explicitly represented within the ModSAF Compact Terrain Database (CTDB) data structures. Soil types are being

expanded as appropriate to provide meaningful values for the sea floor. The sea surface representation is being handled as a second surface feature overlying the sea floor. Tidal height variations are also supported. Ocean attributes will be dynamically updated with data made available through interaction with other Synthetic Environment programs, namely the Weather in DIS/Total Atmosphere and Ocean System (WINDS/TAOS) and Dynamic Virtual Worlds (DVW), and derived from authoritative sources, as available, including the Master Environmental Library.

Additional support for the representation of the surf zone is also being provided. This includes support of a triangulated irregular network for the coastline, with use of higher resolution hydrography and bathymetry data than may be available in the deeper ocean. This will facilitate modeling the transition from land to sea, without large elevation discrepancies along the shore. ICTDB also supports surf zone attributes of surf height and water temperature, and man-made features in the surf zone, such as jetties, oil rigs, breakwaters, wharves, and piers.

Future work includes expanding the ICTDB representation to provide a higher fidelity ocean and surf zone representation. The ocean will be treated as a number of volumetric features with similar attribution. New surface and subsurface features will be added to represent more of the ocean characteristics. More explicit modeling of the surf zone will be provided. Ocean subfloor characteristics will also be represented.

2. Design

In order to support Navy and Marine Synthetic Forces simulations (Tracor, 1996), the Compact Terrain Database (CTDB) used by ModSAF has been expanded to include a representation of the ocean floor along with a more complete representation of the ocean surface. Table 1 shows the various ocean characteristics that are represented.

The ocean bottom is represented using the existing terrain representation (i.e., grids, TINs, and microterrain) (Stanzione, et. al. 1996), and additional supported soil types. Many ocean “features” are really abstractions describing pieces of the terrain. As such, the physical representation of such features can be adequately handled by incorporating their structure into the polygonal representation of the ocean bottom. Abstract notions such as “this area of the terrain is a reef” can be explicitly stored as abstract features using existing CTDB mechanisms.

The representation supports tidal variation of the ocean surface. In the coastal regions, the absolute elevation of the water’s surface is specified, subject to some maximum x-y bounds. Within the specified region, any area where the water elevation exceeds the land elevation is covered by water, and any area where

this is not the case is dry (or perhaps moist) land, as shown in Figure 1. The bounding polygons for water bodies are defined by high tide position, so that the surface elevation can be decreased to represent lower tide levels. In order to allow for changing tides in real-time, all water polygons reference a tidal zone. Each zone stores an offset, which is added to the surface elevation stored for the polygon. Thus, changing the tide in a region is simply a matter of changing that region’s tidal offset.

In most areas, the ocean surface is represented by single square polygons that correspond to the size of a CTDB terrain patch. The representation can not be too coarse because in databases that use the Global Coordinate System (GCS) (Evans, 1995) the ocean surface is curved. On the other hand, the representation can not be too fine or it will use much more memory. The patch size is the largest size at which integration into existing intervisibility algorithms is straight forward, since the intervisibility code already performs a patch traversal. The representation consists of a single elevation value for water in the patch, and a reference to additional surface characteristic data. It is assumed that there will be few unique sets of surface characteristics relative to the number of patches.

Table 1: Ocean Representation Characteristics

	Multiple Elevation Surfaces	Advanced Features and Attributes	Dynamic Terrain
Ocean Floor	<ul style="list-style-type: none"> ◆ Bathymetry data ◆ Extended soil types infrastructure to include bottom characteristics 		
Ocean Surface	<ul style="list-style-type: none"> ◆ Patch and Wet TIN surface ◆ Sea State attributes (primary and secondary wave height, period, speed, direction) ◆ Surface Temperature 		<ul style="list-style-type: none"> ◆ Dynamic sea state and surface temperature
Surf Zone	<ul style="list-style-type: none"> ◆ Tidal Zone with offset for surf height 	<ul style="list-style-type: none"> ◆ Man made features (wharves, piers, etc.) 	<ul style="list-style-type: none"> ◆ Variable tidal zone offset
Rivers	<ul style="list-style-type: none"> ◆ Wet TIN surface 		



Figure 1: Tidal Variation

For areas where patches are simply too big, such as along the coastlines, a triangulated irregular network (TIN) of polygons is used to represent the surface. These “wet TINs” contain elevation data, as well as a characteristic reference and tidal zone as described above. This representation is also used to represent most river surfaces.

For areas where multiple water polygons overlap, the highest one is assumed to be correct. Consider the case shown in Figure 1. At high tide, the water level is above the ridge in the middle, while at low tide it is below it. As the tide recedes, water is trapped beyond the ridge. Thus, when the tide is below the level of the ridge, the water level to the right of the ridge will stay at the level of the ridge, but when the tide is higher, the water level everywhere will be the tidal level. We can handle this by placing additional wet TIN polygons to the right of the ridge at the level of the ridge, and also having a tidal polygon that covers that region.

A number of existing CTDB data structures were modified to support this representation. A flag was added to the patch group header for each patch specifying whether that patch represented a water surface. Within each patch data structure, a water elevation field was added, which is only valid if the patch water flag is set in the header. Also, pointers were added for soil type and water characteristic attributes.

To support the additional soil types needed for the ocean bottom, we have added a level of indirection by

storing a soil table reference per patch or patch group. Most soil tables have 16 entries, since microterrain and grid posts only support 4 bit soils. However, the CTDB TINs representation supports 8 bit soils, so 256 entry soil tables are supported as well, allowing greater flexibility for TINed regions.

An additional microterrain type was added in order to support the wet TINs. A characteristic reference word was added to the microterrain data structure in CTDB, which points to a new data structure that contains the water characteristics for the wet TIN polygons. These water characteristics include sea state, temperature, and tidal zone index, which is an index into an array of tidal zones. The tidal zone reference was put in with the other characteristics, since it is expected to be constant over large regions. A reference was used rather than storing the data directly to minimize the number of places in which tidal data actually resides, since this is something that may be modified frequently at run-time, and localization is critical.

3. ModSAF Modifications

The scope of the ModSAF modifications was limited to the terrain database library libCTDB. The general design principle was to add a representation that would support tidal variant water surfaces and support the basic terrain utilities, intervisibility and elevation lookup, to function above and below the water surface.

3.1 Variable Water Surface

A variable water surface was added to ModSAF's libCTDB to allow modeling of the dynamic ocean surface for Naval and Littoral operations. The first phase of development, integrated into ModSAF 2.1, was to add the infrastructure to support water surfaces and ocean bathymetry. The bathymetry was integrated into the original TIN topology and the water surface, added via patch water or wet TINs, was placed at the mean high tide level and a tidal attribute was referenced to determine the actual tide level of the water surface. In any regions where water anomalies may occur (e.g. captured water) the water surface representation was augmented to allow this region to contain multiple water surfaces each referencing a different set of tidal attributes.

3.2 Elevation Engine

The "elevation engine" is the core utility behind all vehicle placements and soil and elevation queries. Prior to variable surfaces, the terrain database optimized its elevation query scheme by enforcing a hierarchy of features classes and subclasses. By knowing that one feature class always superseded the other in elevation, the elevation lookup processing could terminate once the first feature surface was intersected. For example, if both a canopy and building reside at the same (x,y) location, then the building elevation is always returned, even if the canopy elevation is above the building.

With the addition of dynamic elevation features (ocean surface) and multi-elevation surfaces (Stanzione, et. al, 1996 (2)) the database can no longer guarantee that the first intersection found is the intersection of interest. This requires that the elevation engine acquire information about all feature surface intersections before returning the correct elevation. For example, if the water surface is modeled via patch water (separate feature class from terrain), then a point (x,y) that resides in the littoral region would be interested in the terrain elevation at low tide (assuming that the water revealed the terrain) and the water surface at high tide. This determination would be possible only after all terrain surfaces (grid and TIN) and all water surfaces (patch water and wet TIN) were inspected.

Requiring that all features and terrain at a point (x,y) be included in the elevation query increases the

amount of time to find an elevation. In order to limit this additional processing, two new elevation query routines were added to libCTDB to allow more targeted elevation queries and thereby allow the user to reduce the types of features inspected. It is important to note that some profiling tests were performed on the standard elevation lookup routine to determine if the changes made for multiple elevation queries significantly affected its performance. The result was that the performance was slightly degraded, by less than 6%. It was determined that this was an acceptable expense for the improvement to the elevation lookup functionality.

3.2.1 ctdb lookup qual elevation

This routine is a hybrid of the following routines:

```
ctdb_lookup_elevation(_ml)
ctdb_lookup_soil(_ml)
ctdb_lookup_max_elevation
```

These routines look for a single elevation that is the maximum or the closest surface at or below the input reference surface. Ctdb_lookup_qual_elevation pulls all of the above functionality together into one function by the addition of an elevation qualifier and an elevation data structure. The elevation qualifier is an enumeration that allows control over the feature classes (e.g. volume, linear, canopy) and terrain classes to include in the elevation lookup (e.g. land, water), and surface information (material/soil type) to be returned. Table 2 shows the defined elevation qualifier values.

Ctdb_lookup_qual_elevation can also be used to determine if a feature class or terrain class exists at a certain point. For example, the libCTDB routine ctdb_point_within_canopy could be replaced with the following call:

```
ctdb_lookup_qual_elevation(ctdb, x, y,
                           (CTDB_INCL_CANOPY |
                            CTDB_RETURN_MAX),
                           0, CTDB_ELEV_DATA *)
```

This again stresses the new flexibility added to the core elevation lookup functionality.

In addition to the qualifier interface to elevation lookup, the user has the option of getting comprehensive information about the terrain intersection through a new output data structure CTDB_ELEV_DATA. If a non-NULL address is

Table 2: Defined Elevation Qualifier Values

CTDB_USE_DEFAULT	Same result as ctdb_lookup_elevation
CTDB_INCL_MICRO_ML	Include multi-level microterrain in the search
CTDB_INCL_VOLUME	Include all volumes in the search (e.g. buildings, multi-elevation structures, etc.)
CTDB_INCL_LINEAR	Include all linears in the search (e.g. treelines, dragon teeth, etc.)
CTDB_INCL_CANOPY	Include all canopies in the search, not including treelines
CTDB_INCL_LAID_LINEAR	Include laid linears in the search (e.g. roads, rivers, etc.)
CTDB_INCL_MATERIAL	Include the surface material in the search (i.e. soil)
CTDB_INCL_TERRAIN_SKIN	Include the terrain skin (grid and or TIN surfaces)
CTDB_INCL_MICRO_DEF_BASE	Include default or base microterrain in the search
CTDB_INCL_MICRO_WATER	Include water microterrain in the search
CTDB_INCL_VEHICLES	Include individual vehicles in the search
CTDB_INCL_SINGLE_TREES	Include individual tree models in the search
CTDB_INCL_PATCH_WATER	Include patch water in the search
CTDB_RETURN_MAX	Return only maximum elevation
CTDB_RETURN_MIN	Return only minimum elevation
CTDB_INCL_WATER	Include anything that is considered to represent water surfaces
CTDB_INCL_MICRO	Include any type of microterrain (default, multi-level, base and water)
CTDB_INCL_LAND	Include all elements in the terrain database that are considered to constitute the terrain surface (terrain skin)
CTDB_ALL_FEATURES	Include all classes of features in the search
CTDB_LAND_AND_WATER	Include all land and water
CTDB_MAX_LAND_AND_WATER	Include all land and water and return only the maximum
CTDB_MAX_LAND_NO_WATER	Include all land and return only the maximum

provided, the elevation routine will populate the data structure with the elevation, feature class, subclass and the surface information, as shown in Table 3.

Table 3: Elevation Data Structure

Z	Elevation
Class	Terrain class (e.g. Linear, Volume, etc.)
Subclass	Terrain subclass (e.g. Terrain Skin, Building, Treeline, etc.)
Material	FACC Material Attribute Code
Normal Vector (X, Y, Z)	Normal vector of elevation surface

3.2.2 ctdb lookup elevation mes

This new CTDB function is similar to the ctdb_lookup_qual_elevation routine but returns all of the intersected elevation surfaces, as specified by the input qualifier. The output of this routine is an array of CTDB_ELEV_DATA data structures. The list is

terminated by a feature class of CTDB_FC_ILLEGAL which denotes an illegal feature class.

3.3 Intervisibility Engine

LibCTDB uses an “intervisibility engine” to perform a number of tasks. The intervisibility engine implements a linear traversal along the terrain surface, noting all terrain and feature edges crossed. This functionality is used to implement line of sight calculations, generation of terrain profile vectors, and high ground calculations along a linear path.

Prior to the addition of the ocean representation, an intervisibility query always assumed that terrain blockage was from below. To allow intervisibility from below the water surface, the ModSAF intervisibility algorithm needed to be modified to support blockage from below and above the terrain, possibly simultaneously. Depending on which intervisibility algorithm is used (profile, high ground, ground intersection, etc.), the desired effect of water surfaces may be different. For the profile vector the

user may wish to know all terrain edge crossings without taking the water surface into account (e.g. when implementing a submarine behavior), but a tank behavior would always take the combination of land and water into account. Greater control has been added to the intervisibility routines through the passing of an elevation qualifier to provide more control over the features to be taken into account.

Currently, as shown in Figure 2, the ocean surface always blocks intervisibility from either above or below. The ocean volume has the same affect on intervisibility as air. It is anticipated that the intervisibility through the ocean volume, as well as the atmosphere, will be modified by environmental models utilizing the WINDS/TAOS atmospheric and ocean databases, when this system is available (Whitney, 1996).

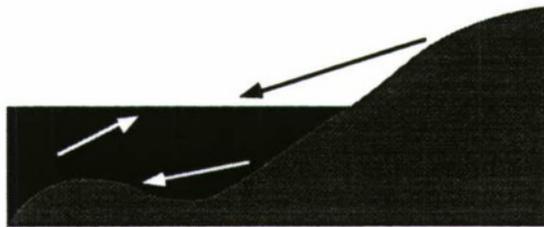


Figure 2: Ocean Intervisibility

3.4 Ocean Attributes

As mentioned above, the dynamic water surface is managed through a tidal attribute. By changing the tidal attribute, all of the water mapped to that

attribute is modified. In addition to the tidal attribute, the other ocean attributes added to ModSAF 2.1 include surf height, temperature, and primary and secondary wave characteristics, which are period, speed, amplitude and direction.

In order to access or modify these attributes, a simple read and write interface was added to the libCTDB API, with expectations to enhance the functionality in future releases. The point read and write routines contain variable list interfaces which allow easy attribute access (single or multiple in a single query) and attribute expandability without requiring a change to the published API.

3.5 Global Coordinate System

To support the global coordinate system, the patch water processing was modified to account for the curvature that occurs at the extreme edges of cell databases. With flat patch water, where each patch uses a single elevation for the water surface in the patch, potentially large step discontinuities could exist between patches, as shown in Figure 3. To better model the water curvature and facilitate tidal variation, the patch elevation is mathematically modeled using the WGS-84 ellipsoid. The tidal variation is applied to the ellipsoidal elevation to determine the true water elevation. To optimize the computational load when determining the WGS-84 ellipsoidal elevation, intermediate results for a specific GCS cell are cached. Both the elevation lookup and intervisibility engines were modified to lookup the adjusted elevation when analyzing surfaces or edge intersections.

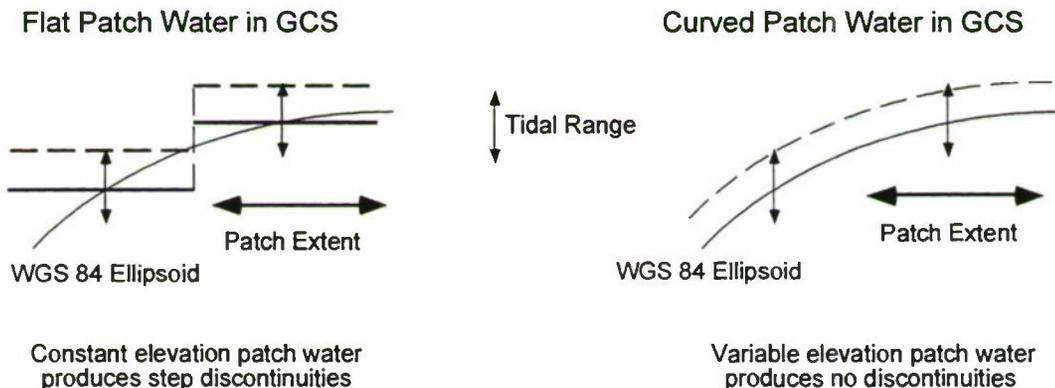


Figure 3: Patch Water Elevations in GCS

4. Compiler Modifications

In order to generate databases using the ICTDB ocean representation, modifications were made to both of the existing CTDB database compilers. The **recompile** program is used to convert old format CTDB databases to newer formats and vice versa, and also supports addition of user specified terrain data to an existing database. This program was modified to optionally generate synthetic bathymetry in water covered areas and to allow user input of multi-level water features. The **slkctdb** program generates a CTDB format database from S1000 data. This program was also modified to generate synthetic bathymetry data, as well as correctly compile multi-level source data when available. These modifications allowed us to successfully generate databases making full use of the multi-level ocean representation, and in the process highlighted some deficiencies in current source data representations.

Since very few datasets exist that include both bathymetric and surface data in formats that can be easily converted to CTDB, it is important to allow compile time modification of existing datasets to include synthetic bathymetry data. This facilitates testing both by developers and by the user community. For this purpose, a relatively simple conversion mechanism is adequate. With this in mind, we chose to duplicate each water polygon found in an existing database, "push" it down a user-specified distance, and change its soil type, typically to "sandy". This generates an ocean "bottom" with constant depth. While this does result in discontinuities at the ocean-shore boundary and hence does not allow for realistic surf zone simulation, the result is adequate for simple deep water usage. This mechanism has been successfully implemented in both the **recompile** and **slkctdb** programs. We have experimented with more sophisticated bathymetry generation algorithms that attempt to generate more bowl shaped ocean bottoms with some success, but have not attempted to perfect them as we believe that users who require realistic bathymetric data would be better served by using real world datasets as they become available (see discussion of the experimental Camp Pendleton dataset below).

In addition to supporting the generation of synthetic bathymetry data, the **recompile** program has been modified to allow users to add multi-level ocean data by hand. This is supported via three mechanisms, all

of which read data from user-generated ASCII files at database compilation time. The first allows the user to add microterrain to a database by specifying a list of terrain triangles to add. This allows the user to replace the existing flat ocean data with a representation of the ocean bottom that is as complex and accurate as needed. The second mechanism allows the addition of patch water over broad areas or in specific patches. This allows rapid generation of ocean surface data in regions of deep water. Finally, the user may also specify wet TIN polygons as a list of triangles, allowing detailed specification of the water surface in the surf zone or other areas where patch water polygons are too coarse. Together, these mechanisms support the addition of multi-level ocean data to support each user's needs without requiring changes or additions to existing source datasets.

The mechanisms described above allow a great deal of flexibility. However, it is important that CTDB also support the use of multi-level ocean data in existing source formats where available. During the course of ocean representation development, we were able to obtain a preliminary S1000 dataset for the Camp Pendleton region that included explicit polygonalization of both the ocean surface and the ocean bottom. Using this dataset for testing, we modified the **slkctdb** program to correctly handle source data that includes both bathymetric data and ocean surface data. To take advantage of the efficiency of the patch water representation, water surface polygons are converted to patch water wherever possible. In cases where this cannot be done, for example in patches that are only partially covered by water, wet TINs are used.

While we were able to successfully generate a useable CTDB format database from multi-level S1000 source, we did encounter some difficulties. Like CTDB, S1000 has historically been used to represent terrain with only a single surface at any given (x,y) point. As such, it provides only limited support for the types of queries a multi-level CTDB compiler needs to make. For example, the CTDB ocean representation treats water surface polygons which have corresponding bottom data below them fundamentally differently than "standalone" water polygons which do not. Thus, it is important to be able to query each polygon to determine whether or not there exist other polygons covering the same area but representing other surfaces, a query which is not currently supported. Similarly, S1000 provides an elevation lookup query that returns a single elevation at a given (x,y). As discussed earlier, we found it

necessary to augment such interfaces in CTDB, and believe that this should be done for other formats as well.

5. Future Work

The work that has been done to date has been to develop an initial implementation of an ocean representation. This representation can be further expanded to support a more realistic ocean representation by adding the characteristics shown in Table 4. These include more dynamic characteristics, as well as additional features and attributes. A significant addition over the initial implementation would be the representation for features and surfaces within the ocean volume, such as areas of high turbidity and thermal layers. The boundaries of these

features, as well as other attributes, could be obtained from the TAOS ocean database, which is being developed as part of the WINDS/TAOS program.

6. Conclusion

The ocean representation described in this paper has been integrated into ModSAF 2.1, including the API routines to change the ocean attributes and the compiler modifications to generate databases from source data with and without bathymetry data. The ocean representation is compatible with all of the coordinate systems currently in use within ModSAF, including GCS. Future work will expand this representation to produce an even higher fidelity ocean representation for use by a variety of CGF behavior developers.

Table 4: Future Ocean Representation Characteristics

	Static Representation	Dynamic Representation
Ocean Floor	◆ Subfloor characteristics	◆ Dynamic subfloor characteristics
Ocean Surface	◆ Expanded sea state (salinity, turbidity) ◆ Currents ◆ Ice	◆ Dynamic expanded sea state, currents, and ice
Ocean Subsurface	◆ Volumetric features (temperature, density, salinity, turbidity) ◆ Underwater surfaces (thermal layer)	◆ Dynamic volumetric features and surfaces
Surf Zone	◆ Natural features (reefs, rocks) ◆ Mobility characteristics	◆ Variable boundaries of features ◆ Dynamic mobility characteristics (possibly breaking waves)
Rivers		◆ Variable height for flooding

7. Acknowledgment

This work is being done as part of contract DACA76-94-C-0022 from the Defense Advanced Research Projects Agency (DARPA) and the US Army Topographic Engineering Center (TEC). The authors wish to thank George Lukes of ARPA and Kevin Mullane of TEC for their interest, encouragement, and guidance.

8. References

- Evans, A., Stanzione, T. (1995), "Coordinate Representations for CGF Systems", 13th Workshop on Standards for the Interoperability of Distributed Simulations.
- Stanzione, T., Braudaway, W., Chamberlain, F., Drutman, C., Roberts, I., Evans, A., Buettner, C., Lu, H., D'Urso, R. (1996), "Integrated CGF Terrain Database Interim Technical Report", TASC.
- Stanzione, T., Chamberlain, F., Mabijs, L., Sousa, M., Evans, A., Buettner, C., Fisher, J., Lu, H. (1996), "Multiple Elevation Structures in the Improved Computer Generated Forces Terrain Database", 6th Computer Generated Forces and Behavioral Representation Conference.
- Tracor Applied Sciences (1996), "Environmental Modeling Requirements for JCOS Maritime Systems and Forces", Doc. Number T95-01-9589-U.
- Whitney, D., Reynolds, R., Schmidt, E., Driscoll, M., Olson, S. (1996), "Integrated Ocean-Atmosphere Environmental Data Services for Distributed Simulations", 14th Workshop on Standards for the Interoperability of Distributed Simulations.

9. Authors' Biographies

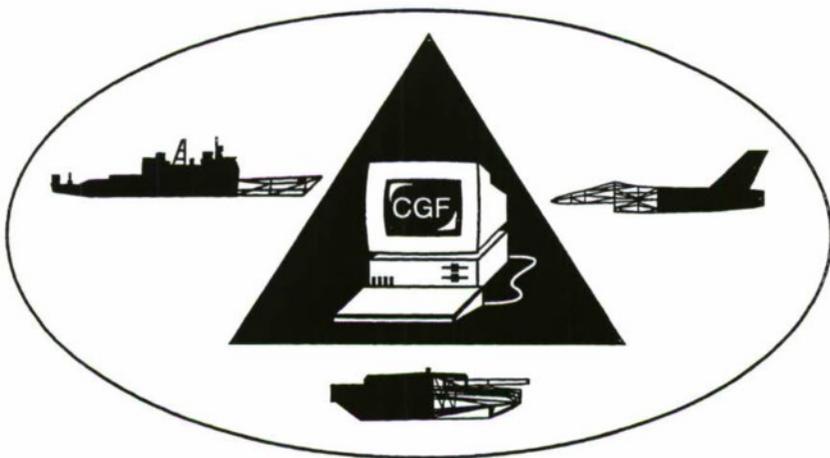
Thomas Stanzione is the manager of the Computer Generated Forces Section at TASC. He is the Program Manager for the ICTDB project and a key contributor to TASC's other CGF programs, including the DIS Exercise Construction Toolset (DISECT) being developed for STRICOM. His interests include data representations for terrain reasoning and terrain database generation for simulation applications. Mr. Stanzione has a Masters

of Science degree in Photographic Science from the Rochester Institute of Technology.

Alan B. Evans Dr. Alan Evans is the manager of the Burlington office of SAIC and is the technical lead on the ICTDB project. He has worked in Advanced Distributed Simulation for over five years with primary research interests in simulation systems architecture, object modeling, distributed object technology and synthetic environments. Dr. Evans has a Ph.D. in Mathematics from Michigan State University and an M.S. in Computer Science from New York University.

Cedric Buettner is a Senior Software Engineer at SAIC, Burlington. He has been involved in the integrated TIN representation, Global Coordinate System, Ocean Representation and the Multiple Elevation Surfaces development under the ICTDB program. Prior to joining SAIC, he worked on Raytheon's Patriot Fire Unit software developing prototype tactical system enhancements. He is a graduate of Gordon College in Wenham, MA with a BS in Physics and Mathematics.

Forrest Chamberlain is a Member of the Technical Staff in the Computer Generated Forces section at TASC. Forrest has been involved in Computer Generated Forces work since joining TASC in 1994. He is currently responsible for the terrain reasoning and mission tracking components of the Command Forces (CFOR) project at TASC and is a critical contributor to the ICTDB terrain representation effort. Prior to joining TASC, Forrest participated in the hardware and software design of a "wearable" computer system at Carnegie Mellon University, where he earned his Masters Degree in Electrical and Computer Engineering. Forrest earned his BS in Electrical Engineering at Cornell University.



Global Coordinate System in the Improved Computer Generated Forces Terrain Database

Thomas Stanzione
Forrest Chamberlain

TASC
55 Walkers Brook Drive
Reading, MA 01867
tstanzione@tasc.com
flchamberlain@tasc.com

Dr. Alan Evans
Cedric Buettner
Howard Lu

SAIC
Suite 130
20 Burlington Mall Road
Burlington, MA 01803
aevans@bos.saic.com
buettner@bos.saic.com
hlu@bos.saic.com

1. Abstract

The Improved Computer Generated Forces Terrain Database (ICTDB) project, being developed jointly by TASC and SAIC, is one of four projects in the DARPA/TEC Advanced Distributed Simulation Synthetic Environments program. The goal of this project is to design and develop the next generation terrain database representation for Computer Generated Forces (CGF) systems. One of the major technology developments undertaken by the ICTDB team is the design and implementation of a Global Coordinate System (GCS) for CGF systems. GCS starts with a tiling of the Earth's surface into cells compatible with the GEOREF tiling. Within each cell, a local Cartesian frame of reference is defined whose axes furnish a secant plane to the Earth's surface and a normal vector at the center of the cell. This amounts to an offset and a rotation of the DIS GCC coordinate system. The benefits of moving to GCS include: elimination of projection anomalies, naturality to the application developer, fast conversion to and from GCC and true global scaleability. At the same time, the compactness of the data representation is maintained. A more complete description of the design of GCS is contained in a paper by Evans and Stanzione presented to the 13th DIS Workshop.

In the current paper, the details of the ModSAF implementation of GCS are elaborated. To date, the implementation of GCS exists in ModSAF, as additions to the CTDB representation, together with several new libraries to support the tiling scheme and coordinate translations. This GCS framework has in

fact been integrated into ModSAF 2.1, together with changes to the Persistent Object Protocol which support the extra coordinate used by GCS, the cell ID. We then discuss in detail some issues of scaleability which argue for the use of the Global Coordinate System. Our analysis shows that for the upcoming STOW ACTD, use of GCS seems a strong requirement. Finally, we outline future work needed to make the implementation of GCS complete. This includes changes to the database production process needed to fully support GCS. Currently multi-celled GCS databases have been produced from the Area 2 dataset for demonstration purposes.

2. Overview and Implementation Progress

First, we give an overview of the problems addressed by the Global Coordinate System and list the CGF requirements. We also survey the progress to date on design and implementation.

2.1 Overview

In a paper presented to the simulation community at the 13th DIS Workshop, Evans and Stanzione outlined the arguments for switching to a Global Coordinate System (GCS) representation of the synthetic world for internal use by CGF systems (Evans and Stanzione, 1995). This paper dealt primarily with issues relating to the local *faithfulness* of the representation, as well as the space optimizations deemed necessary to achieve acceptable real-time system performance. The current

paper looks more carefully at some of the issues related to *scalability* of the internal CGF model of the synthetic world.

GCS is based on a tiling of the Earth's surface into *cells*. In each cell, a local Cartesian coordinate system is used which is offset and rotated from GCC, Geocentric Cartesian Coordinates. The resulting frame of reference has its origin at the center of the cell, with X and Y pointing North and East, respectively, and the Z axis normal to the Earth's surface at the center of the cell. A location in GCS is four-dimensional, specified by X, Y and Z in the cell frame of reference together with a *cell ID*. In the current paper, we first review the requirements which led to the design of GCS. We then look at progress to date on the GCS implementation, and try to understand why current coordinate representations of the synthetic world are inadequate to achieve the objectives of STOW, as well as future programs. We close with a series of recommendations on what should be done next to complete the implementation of GCS in the ModSAF-derived family of CGF models.

2.2 Requirements

The following are the fundamental requirements which argue for the adoption of GCS:

- **Scaleability** - As simulations grow larger in scale, it is crucial that a representation be used internally which supports arbitrarily large exercise areas, perhaps even the entire surface of the Earth.
- **Compactness** - Storage must make efficient use of storage space. This requirement is, of course, closely related to the scaleability requirement.
- **Faithfulness** - The coordinate representation should be free of anomalies such as curvature effects.
- **Ease of translation to and from GCC** - For efficiency reasons, an internal coordinate representation should support fast conversion both to and from GCC, since this translation must occur for every location vector which is read from or written to the network in a DIS exercise.
- **Naturality** - The coordinates values returned to software components simulating platforms and

command elements must be natural in the sense that they must have an intuitive relationship to the real world for the benefit of developers of this code.

In the paper cited above by Evans and Stanzione, it was shown how GCS was designed to satisfy these requirements.

2.3 ICTDB Implementation Progress

An implementation of the Global Coordinate System now exists in ModSAF. The fundamental idea is to produce one CTDB database per cell, with some overlap at the edges to insure consistency. Over the past four months, framework libraries have been integrated into ModSAF to support GCS. The additions to ModSAF include two new libraries: *libgcs* and *libworld*. *Libgcs* provides the inter-cell coordinate transformations required by *libcoordinates*, as well as the 4D vector algebra required by the application, while *libworld* implements the tiling scheme and handles initialization of the playbox in GCS. Also, modifications have been made to *libctdb* which make the intervisibility code aware of cell transitions. In essence, when a line of sight (LOS) crosses cell boundaries, the intervis engine is invoked recursively using the features and elevation data of the new cell. In actuality, any LOS used in simulation in the near-ground environment would not intersect more than three cells, since LOS calculations are typically much shorter than the cell size of 100 km. This extreme case could occur at a vertex joining four cells. Eventually as simulations of imaging radar and other high-flying sensors are incorporated, LOS calculations involving many cells may occur. Finally, changes have been made to the Persistent Object Protocol to incorporate the cell ID required by GCS into all PO data structures which contain positional information.

Furthermore, several Proof Of Principle demonstrations have illustrated the use of the framework modifications. The most important demonstration was a depth-first modification of the slice of behavioral and planning support routines necessary to fly simple FWA missions on multi-cell GCS databases. None of the changes in behavioral libraries needed to make the application GCS aware have as yet been checked in to baseline ModSAF code.

Finally, when it comes to generating GCS databases, the tiling algorithms have been added to the ModSAF compiler which takes S1000 source data and produces CTDB format data. This process can be switched at run-time to either output a true multi-cell GCS product or a single-cell product which contains GCS. data. The latter is reminiscent of earlier "curved Earth" databases used in experiments and trade shows.

3. Issues of Coherence and Scale

In this section, the core issues related to scalability and coherence of the GCS cell framework are examined. Specific attention is paid to the system requirements of the STOW ACTD.

3.1 Vertical Deflection

GCS solves a fundamental problem inherent with projected coordinate systems. It restores the local faithfulness of a real-world coordinate system. In other words, straight lines in a GCS cell frame of reference correspond to straight lines in the real world. However, there is a trade-off which leads to another kind of distortion as the extents of the playbox increase. In a single cell, the Z axis is normal to the Earth's surface at the center of the cell. Moving away from the center of the cell, the *vertical deflection*, or the angle between increasing Z and the normal to the Earth's surface, becomes non-zero. For a cell of 100 km by 100 km, this vertical deflection would be about 0.7 degrees at the corners of the cell. Current plans call for a playbox in STOW 97 of 5.5 degrees by 7.0 degrees. This means that for a single cell GCS database, the vertical deflection at the periphery of the playbox would grow to between 2.5 and 3.5 degrees, which could begin to be significant for ground platform models. While the vertical deflection is provided to the application by *libgcs*, modifying all the ModSAF ground platform models to account for the vertical deflection is not appealing. Thus, to limit the size of the vertical deflection as databases increase in scale, multi-cell GCS is necessary. We should add that for a playbox of this size, it has long been recognized that UTM databases are unacceptable. This is due to compatibility problems in spanning multiple map zones, datum inconsistencies and the slowness of coordinate conversion from one projected UTM frame of reference to another.

3.2 Effective Slope

In a GCS database, the values of Z on the reference ellipsoid decrease as one moves away from the center of a cell. This decrease is the result of the curvature of the Earth, and is a natural corollary of the fact that GCS is faithful to the 3D geometry of the real world. However, the platform models of current CGF systems are calibrated in a local vehicle frame of reference. Relative to his coordinate system, the terrain in a GCS database will appear to be sloped. This effect increases towards the cell periphery. Assume that cells are about 100 kilometers square and that the cell is subdivided into *patches*, with each patch 500 meters square. A simple calculation with the equation of the WGS 84 ellipsoid shows a decrease in Z values of between four to six meters, moving from the inner patch boundary to the outer patch boundary for patches around the periphery of a cell. On average, this means that the slope of the ground in such patches would be about 0.6 degrees, exclusive of local terrain morphology, or that driving towards the cell periphery, one would appear to be going slightly downhill. We emphasize that this effect is relative to vehicle frame of reference. The underlying GCS cell data correctly model the Earth's curvature. Our estimate is that this "effective slope" would have no significant impact on ground vehicle dynamics for single-cell GCS databases of 100 km square or less. However, the effective slope would increase proportionately as the size of the cell increased to as much as two degrees on a database with the extents expected in STOW 97. This would begin to affect mobility calculations based on defragmented slope values, and may lead to unacceptable changes in simulation behaviors.

3.3 Accuracy Limits

While the effects of vertical deflection and effective slope described in the previous two sections may be important, a more convincing argument for using multi-cell GCS databases in STOW 97 revolves around the implementation in ModSAF's CTDB. In CTDB, elevation values are stored using a fixed point basis which reserves twenty-one bits for Z values. As is well known, this leads to an accuracy greater than a centimeter over a vertical range of 10,000 meters. However, a simple calculation with the equation of the WGS 84 ellipsoid shows that in a GCS database with cells of 100 km by 100 km (approximately one degree by one degree), the range of Z values will at least 500 meters from the center of the cell to the cell

boundary, which would be about 70 km distant at the corner of the cell. If the size of a cell increases to five degrees or more (350 km from cell center to the corner), as is anticipated in STOW 97, over a quarter of the range of Z values at the standard accuracy in CTDB will be used in modeling the curvature of the Earth alone. When surface morphology is added in mountainous areas of the world, we see that the storage capacity of the ModSAF CTDB will be strained, forcing a decrease in accuracy. The only alternative would be to restructure the bit fields used to store elevation values. This would be an unappealing alternative from an engineering standpoint. The obvious conclusion is that multi-cell GCS databases are necessary to support exercises on the geographic scale of STOW 97. One might argue that this conclusion would be invalid if appropriate modifications were made to the CTDB representation. However, as exercise scale increases further in the future, it is safe to assume that any modest increase in accuracy so achieved would eventually be made obsolete.

3.4 2D Anomalies and the User Interface

With the integration of GCS into the ModSAF framework, an important change occurred in the way the synthetic battlefield is presented to the user. Previously, the Plan View Display (PVD) used the internal UTM or SIMNET coordinates, converted to MGRS (Military Grid Reference System) to render a view very much like an actual map projection on the workstation desktop. With GCS, cell coordinates in a single-cell database are mapped directly to screen coordinates. In a multi-cell database, all coordinates are transformed to the frame of reference of the cell containing the point in the center of the screen, then converted to screen coordinates. The overall effect is as if the user were looking down at the Earth's surface from an aerial viewpoint, instead of looking at a map. MGRS grid lines are rendered on top of the map, and at very high zoom levels, the lines will appear to be slightly curved.

The fact that GCS models the curvature of the Earth means that at high zoom levels, some behaviors of the ModSAF GUI have changed. An areal feature which maps to an image of U square units in screen coordinates, when rendered at the center of the PVD, will take up less than U square units when it appears towards the edge of the window, and will actually appear smaller to the human viewer. This is of course what the human eye would see when looking down at

the Earth from above. It is not what you would see when panning across a map with its projected view of the world. This foreshortening does have some effect on the software which does high-level motion planning. Since the planning libraries such as *libroutemap* reason about 2D locations only, they are essentially working with the projection of cultural and terrain features onto the GCS secant plane. Relative locations of features are preserved, but the absolute sizes of derived features, such as mobility corridors and avenues of approach, are decreased slightly towards the periphery of a cell. Our estimate is that such effects are negligible, but more quantitative study would be useful.

In addition to the intra-cell 2D concerns cited above, there are inter-cell problems when using 2D data. For motion planning, a 2D location (X, Y) is really a line parameterized by Z . When a transition is made to an adjacent cell, it is not obvious what the corresponding 2D location (X', Y') should be. If the point $(X, Y, 0)$ is simply transformed to the adjacent frame, the resulting point (X', Y', Z') will have Z' non-zero. Projecting onto the secant plane will give a 2D location in the new secant plane which is actually at a different location in the database. This is in fact what *libcoordinates* does now. This 2D inter-cell coordinate conversion is used on routes spanning cell boundaries, but is subject to the same small anomalies discussed above. After a number of experiments, the ICTDB team came to the conclusion that these anomalies could be ignored. In general 2D conversions must be applied often to PO database objects near cell boundaries, since the PO protocol now supports the GCS tiling in addition to (X, Y) but does not support Z .

4. Future Work

The fundamental technical problem of designing and building a Global Coordinate System which satisfies the requirements of Section 2.2 has been solved. The full realization of this part of the STOW 97 system still requires significant engineering efforts. Looking to the future, the DoD goal is to build unified SAF architecture. We briefly discuss the following four areas:

- Full ModSAF GCS awareness
- Service SAFs and GCS
- Database production process and source data
- SAF Integration

4.1 Full ModSAF GCS awareness

In order to make ModSAF GCS-aware, all of the behavior and platform model libraries will be affected. This integration task has been estimated by the ICTDB team to require on the order of 12 staff-months of effort for the existing models in the Army SAF baseline. Some initial design work on adding a Relative Coordinate System (RCS) for ModSAF platform models has already taken place. RCS will support highly localized frames of reference, down to the level of vehicle components and articulated parts. The intention is to add support for RCS and GCS at the same time. Some further design work on RCS is needed. The OpenSAF GCS integration is tentatively planned to take place just prior to ED2.

4.2 Service SAFs and GCS

If we assume that multi-cell GCS becomes the database framework for STOW 97, then the Service SAF models (AF SAF, FastFleet, MC SAF) will need to be made GCS-aware as well. This integration task has been estimated to require on the order of six staff-months of effort.

4.3 Database Production Process

The run-time databases used by ModSAF currently are derived from S1000 data. The tiling algorithm necessary to turn a single-cell database into multi-cell GCS has been added to the ModSAF compiler which produces CTDB from S1000. Currently, since S1000 databases are in UTM coordinates, the CTDB compiler is forced to resample the elevation grid in converting to even a single-cell GCS product. However, S1000 is also used to compile run-time databases for IG systems. The implicit grid triangulation in S1000 derivatives means that these other products are non-interoperable with single-cell GCS CTDB. The current compromise is to resample the grid and TIN, using S1000 elevation queries to populate the TIN. The result is interoperable with the non-GCS IG run-time products. However, the increased storage required and decreased performance could be avoided if the UTM projections were removed from the S1000 production process. For STOW compatibility, the consumers of S1000 data will be made GCS-aware. Clearly the tiling algorithm should eventually be integrated into the S1000 production process. Adding the tiling to other compilers of S1000 into run-time formats

would require less initial investment, but would lead to duplicative, hence less maintainable code.

4.4 CCTT and SAF Integration

The CCTT Environmental CSC makes the same assumptions about coordinates that ModSAF did prior to the integration of GCS. A location in the synthetic environment is specified by two independent coordinates, X and Y, together with a third coordinate Z, all in a Cartesian frame of reference. The surface of the Earth is a two-manifold with Z dependent on X and Y for HOT (Height Of Terrain) queries. The CCTT terrain database production process makes the same use of UTM projections as the process which takes DMA and other source data, producing first S1000 and then CTDB in the ModSAF environment. We have discussed above the effort expended to date to add the GCS framework to ModSAF. SAF Integration work is slated to provide a common service for modeling the terrain for ModSAF and CATT SAF later this year. We recommend (1) that this framework be required to support GCS, and (2) that the full integration of the GCS framework into the objective Integrated SAF system be incorporated into SAF Integration planning.

5. Conclusion

The Global Coordinate System framework has been integrated into ModSAF 2.1. A number of technical reasons argue for the use of GCS in STOW. Future work will complete the integration of GCS into ModSAF, its service SAF derivatives and make GCS an integral part of the database production process.

6. Acknowledgment

This work is being done as part of contract DACA76-94-C-0022 from the Defense Advanced Research Projects Agency (DARPA) and the US Army Topographic Engineering Center (TEC). The authors wish to thank George Lukes of DARPA and Kevin Mullane of TEC for their interest, encouragement, and guidance.

7. References

Evans, A., Stanzione, T. (1995), "Coordinate Representations for CGF Systems", 13th Workshop on Standards for the Interoperability of Distributed Simulations.

8. Authors' Biographies

Thomas Stanzione is the manager of the Computer Generated Forces Section at TASC. He is the Program Manager for the ICTDB project and a key contributor to TASC's other CGF programs, including the DIS Exercise Construction Toolset (DISECT) being developed for STRICOM. His interests include data representations for terrain reasoning and terrain database generation for simulation applications. Mr. Stanzione has a Masters of Science degree in Photographic Science from the Rochester Institute of Technology.

Dr. Alan B. Evans is the manager of the Burlington MA branch of SAIC's Technology Research Group (TRG) and has worked in Advanced Distributed Simulation for over five years. Dr. Evans is the technical lead on the ICTDB effort sponsored under the DARPA Synthetic Environments Program. His areas of interest include simulation systems architecture and performance analysis, object modeling, distributed object technology and synthetic environments. Dr. Evans holds a Ph.D. in Mathematics from Michigan State University as well as an M.S. in Computer Science from New York University.

Cedric Buettner is a Senior Software Engineer at SAIC, Burlington. He has been involved in the integrated TIN representation, Global Coordinate System, Ocean Representation and the Multiple Elevation Surfaces development under the ICTDB program. Prior to joining SAIC, he worked on Raytheon's Patriot Fire Unit software developing prototype tactical system enhancements. He is a graduate of Gordon College in Wenham, MA with a BS in Physics and Mathematics.

Forrest Chamberlain is a Member of the Technical Staff in the Signal and Image Technology Division at TASC. Forrest has been involved in Computer Generated Forces work since joining TASC in 1994. Prior to that, he was a critical contributor to the hardware and software design of a "wearable" computer system at Carnegie Mellon University,

where he earned his Masters Degree in Electrical and Computer Engineering.

Howard Lu has been a Software Engineer at SAIC, Burlington since August 1995. He has been a major contributor to the implementation of the GCS framework, and has also been heavily involved in SEDRIS design specification and prototyping. Howard is a graduate of the Massachusetts Institute of Technology, with M.S. in Computer Science.

Session 8b: Advanced Concepts

**Beheshti, University of Houston
Fields, U. S. Army Research Labs
Tambe, USC/ISI
Sapaty, University of Surrey, UK**



An Adaptive Environment Modeling Method Under Uncertainty

Richard A. Aló, Mohsen Beheshti, Andre de Korvin, Chenyi Hu, and Ongard Sirisaengtaksin
Department of Computer and Mathematical Sciences
University of Houston-Downtown
Houston, Texas 77002

1. Abstract

Different models may be appropriate to model a system under conditions of changing environments. In fact, under a fixed environment several models may be appropriate with different degrees of beliefs attached to them. As the environment changes, the degrees of belief change too. Some of the models represent the system's possible malfunctions while other models might be possible models of normal performance. External events take place which may cause a probabilistic change in the environment. In the present work we will like to model this situation and find an estimate of the probability that the system continues to function at any specified point in the evolution of the environment.

2. Introduction

The main purpose of this work is to provide a mechanism for automated commanders of CGF to make, change, and refine the assessments on the opposing forces upon the new information received. This information might be a change in weather, in terrain conditions or in forces. Assessments of opposing forces or more generally of prevailing conditions are of extreme importance and have been studied in numerous works, Handley et. al. (1995), Hille et. al. (1995), Holmes (1995), Pandari et. al. (1995), and Yin et. al. (1995).

More specifically, we will use Fuzzy Sets (see Zadeh, 1968), the Dempster-Shafer Theory of Evidence (see Shafer, 1976), Norton's (1988) work connecting Dempster-Shafer masses to Markov Chains and other works to set a framework for estimating the probability of changing one assessment to another or to stick with the same assessment. Also, we will develop estimates of the probability of going from a fuzzy assessment to a crisp evaluation and this should have a clear impact on what course of action to take as a consequence.

Fuzziness comes in naturally because sensors and/or reports are far from being precise in most battle situations. The Dempster-Shafer Theory of Evidence comes in because we have different sensors to measure different features, naturally generating masses on subsets of possible alternatives. We need to combine the information yielded by the different sensors/reports. A key notion which helps to

develop the above result is the notion of degree of equality between two fuzzy sets.

The emphasis in this work is on the construction of the transition probabilities rather than on decision making. However, we perceive this work as an important first step for decision making with vague or imprecise information in the context of policy making as in Kleyle and de Korvin and in other applications for example, de Korvin et. al.

The concept of defining probabilities on fuzzy sets originated with Zadeh (1965). Klement et. al. (1981) formally define fuzzy probability measures over fuzzy σ -fields, which extend the basic σ -algebra to the fuzzy domain. Smets (1990) also deals with fuzzy probability measures using an axiomatic approach, while Piasecki (1985) further extends the theory to probability measures on "soft σ -fields" using the concept of a weak separation.

In this paper we consider a Markov chain whose states are fuzzy sets defined on some finite state space X . Although an infinite number of fuzzy sets can be defined on X , we consider chains having only a finite number of fuzzy states. Since we restrict our attention to the finite case, we avoid the measure-theoretic problems that can arise when dealing with probabilities on fuzzy sets.

In a recent paper, Kleyle and de Korvin, it has dealt with Markov chains involving fuzzy transition probabilities. Fuzzy transition probabilities arise naturally when the transition from one state to another is described by such mathematically imprecise phrases as very likely, likely, unlikely, etc. Procedures for decision making under this type of uncertainty which combine fuzzy set theory with the classical theory of Markov chains have been proposed by Kleyle and de Korvin. In the present work we in some sense "reverse the process" and describe a method of obtaining crisp transition probabilities when the states themselves are finite fuzzy sets. Fuzzy states are relevant to situations in which, due to ambiguity in the data itself, the exact state of the system cannot be pinpointed with certainty.

Norton (1988) has related the Dempster-Shafer rule of combination to Markov chains whose states are finite crisp (i.e. non-fuzzy) sets. In so doing he is able to construct an expression for computing the transition

probability from one set-state to another. We first present some special cases in which Norton's formula can be extended directly to fuzzy set valued states. We then use the concept of the degree to which two fuzzy sets are equal to provide a general extension. Lastly, we employ a modification of a result due to Smets (1982) to compute transition probabilities from a fuzzy state to a crisp state which is a member of X .

To enhance the readability of this paper, the more technical details are relegated to two appendices. A simple numerical example illustrates the procedure.

3. Background

To make this paper accessible to the reader without a background in fuzzy set theory, we now introduce some of the most basic concepts of fuzzy sets. A fuzzy set A on space X is defined by its membership function

$$A: X \rightarrow [0,1].$$

The membership function is a generalization of the characteristic function of a crisp (i.e. ordinary) set. For each $x \in X$, $A(x)$ denotes the degree to which element x is a member of fuzzy set A . For a crisp set, of course,

$$\begin{cases} 1 & \text{iff } x \in A \\ 0 & \text{iff } x \notin A \end{cases}$$

For fuzzy sets $0 \leq A(x) \leq 1$. Those x 's for which $A(x) > 0$ constitute the support of fuzzy set A . For notational convenience, we do not distinguish between the membership function and the fuzzy set itself. In effect, the membership function is the fuzzy set.

When the domain $X = \{x_1, x_2, \dots, x_n\}$ is finite, we represent fuzzy set A by the notation

$$A = \sum_{1 \leq i \leq n} \alpha_i / x_i$$

where $\alpha_i = A(x_i)$ denotes the degree to which x_i belongs to A .

3.1 Notation

Finite fuzzy sets are sometimes written as ordered pairs $\{(\alpha_1, x_1), (\alpha_2, x_2), \dots, (\alpha_n, x_n)\}$, but we find the above notation more convenient. In this paper the \sum operator will be used in both the usual summation sense as well as to define finite fuzzy sets. The usage will usually be obvious from the context. Whenever possible confusion may occur, as when \sum is used in both senses in the same formula, the explicit usage

will be specified in the text. Also, to avoid further notational confusion, we use the $+$ symbol to denote division and reserve $/$ to separate the membership from the support of a finite fuzzy set.

The operations of fuzzy union, intersection and complementation are defined in terms of the membership functions as follows:

$$(A \vee B)(x) = \text{Max}\{A(x), B(x)\}$$

$$(A \wedge B)(x) = \text{Min}\{A(x), B(x)\}$$

$$\neg A(x) = 1 - A(x)$$

For a more detailed Account of Fuzzy Sets refer to Zadeh (1965), Dubois and Prade (1980), and Klir and Folger (1988).

We conclude this section with a brief discussion of Dempster-Shafer mass functions and the Dempster-Shafer (D-S) rule of combination. The D-S masses can be successfully applied to the problem of object recognition. Let X be a set of objects x_1, x_2, \dots, x_n . Assume that we look at a particular feature F which has possible values f_1, f_2, \dots, f_p . Assume also that sensors report the values of feature F with some uncertainty, e.g., the sensors may indicate that F has value f_1 with probability p_1 and value f_2 with probability p_2 . This assessment generates a natural mass m defined by

$$m(A_1) = p_1, \quad m(A_2) = p_2$$

where A_1 and A_2 are subsets of X ; A_1 corresponding to all objects whose feature F has value f_1 and A_2 corresponding to all objects whose feature F has value f_2 .

We now look at several features F_1, F_2, \dots, F_m . where f_k denotes possible values of F_k . Each sensor (geared to one particular feature) then generates a mass m_t ($1 \leq t \leq m$) on subsets of X . The D-S rule of combination allows us to combine the information yielded by all the sensors. Each mass m_t must have the following properties:

$$(i) \quad m_t(A) \geq 0$$

$$(ii) \quad m_t(\emptyset) = 0$$

$$(iii) \quad \sum_{A \in 2^X} m_t(A) = 1$$

The sets A for which $m_i(A) > 0$ are called the focal elements of m_i . Mass m_1 and m_2 can be combined into one mass m_{12} and can be defined by

$$m_{12}(A) = m_1 \oplus m_2(A)$$

$$= \sum_{B \cap C = A} m_1(B)m_2(C) + \sum_{B \cap C = \emptyset} m_1(B)m_2(C) \quad (1)$$

The focal elements of the composed mass function $m_{12}(\cdot)$ are the intersections of the focal elements of $m_1(\cdot)$ and $m_2(\cdot)$. An improper composition is given by

$$m_{12}^*(A) = m_1 \oplus^* m_2(A)$$

$$= \sum_{B \cap C = A} m_1(B)m_2(C)$$

(2)

which is simply the numerator of (1) and allows positive mass to be assigned to \emptyset .

The D-S rule of combination can be extended to any finite composition,

$$m_{12 \dots n}(\cdot) = m_1 \oplus m_2 \oplus \dots \oplus m_n(\cdot)$$

(3)

by induction. Furthermore, the computation can be simplified by using an improper composition in the intermediate stages, and then defining

$$m_{12 \dots n}(\emptyset) = 0$$

(4a)

$$m_{12 \dots n}(A) = m_{12 \dots n}^*(A) / (1 - m_{12 \dots n}^*(\emptyset)).$$

(4b)

This result has been stated and proved by Norton (1988), Th. 1.

In the process of establishing the link between set-valued Markov chains and the D-S rule of combination, Norton obtains an expression for the transitional probability of going from state A_i to state A_j in terms of the mass function:

$$P(A_i \rightarrow A_j) = \sum_{k: A_i \wedge A_k = A_j} m(A_k)$$

(5)

where the A_k 's are the focal elements of $m(\cdot)$.

We will apply Norton's limit theorems to estimate the probability of changing one assessment when facing enemy forces. The way our mechanism works we may either stay with our original assessment or narrow further our evaluation, making it more specific. We will come to a natural stopping point where we can not narrow down any more on the information from the reports and/or sensors that we have available. It may happen that we deduce the information to an empty set, meaning that the information was inconsistent. The limiting set of information refinement will correspond to an absorbing set in the sense of Markov chain.

4. The Direct Extension

Since reports and information acquired during combat conditions are typically not precise we would like to introduce fuzzy sets as focal masses. For example, consider a large enemy force, we might have a situation which we believe with strength 0.7 that we are facing a large enemy force while believing with strength 0.3 that we have a large enemy force backed up by a large number of tanks. We now introduce the general notations.

Suppose the focal elements of a mass function are fuzzy sets defined on finite space X which consist of the "generating focal elements" A_1, A_2, \dots, A_n and all possible intersections of these focal elements. We find it convenient to index these intersections with subscript notation. That is,

$$A(i_1, i_2, \dots, i_p) = A_{i_1} \wedge A_{i_2} \wedge \dots \wedge A_{i_p}, \quad (6)$$

where $1 \leq i_1 < i_2 < \dots < i_p \leq n$, $p = 2, 3, \dots, n$. There are such focal elements assuming (as we do) that

$$A_1 \wedge A_2 \wedge \dots \wedge A_n \neq \emptyset$$

This last assumption would be rather restrictive for crisp sets, but fuzzy set intersections are non empty unless two (or more) of the focal elements have disjoint supports. To have a convenient terminology for later discussions we refer to focal elements obtained from a p -fold intersection as in eq. (6) to be p th order states.

REMARK: Suppose n mass functions all having the same n fuzzy focal elements are composed by the D-S rule as given in eqs. (1) and (3). This n -fold composition will have focal element having precisely the hierarchical intersection structure described above. Explicit formulas for a mass function defined by (3) and having this hierarchical structure that can be defined as follows: Let

$S_n(j_1, \dots, j_p) = \{(i_1, \dots, i_n) : i_k = j_1 \text{ or } j_2 \text{ or } \dots \text{ or } j_p \text{ for all } k = 1, 2, \dots, n \text{ and each possible value of } i_k \text{ must appear in the } n\text{-tuple at least once}\}$.
Then the mass in the n -fold composition is given by

$$m_{12\dots n}(A(i_1, i_2, \dots, i_n)) = \sum_{(i_1, i_2, \dots, i_n) \in S_n(j_1, \dots, j_p)} \prod_{1 \leq k \leq n+1} m_k(A_{ik})$$

For a proof see Kleyle and de Korvin. However, the direct extension of Norton's eq. (5) to fuzzy focals will hold whenever there are $2^n - 1$ focal elements with the above structure, whether or not the mass function is formed by an n -fold composition.

General formulas for the transitional probabilities defined by eq. (5) with a mass function having the hierarchical intersection structure defined above can be defined as follows: Clearly

$$P(A(i_1, i_2, \dots, i_n) \rightarrow A(j_1, j_2, \dots, j_q)) = 0 \text{ if } q < p$$

If $q \geq p$, we define

$$S(q-p, v) = \{(r_1, r_2, \dots, r_{q-p+v}) :$$

where $q-p$ of the r_i 's are the $q-p$ indices *not equal* to $j_{k_1}, j_{k_2}, \dots, j_{k_p}$ and the other v indices constitute a subset of v of the indices $j_{k_1}, j_{k_2}, \dots, j_{k_p}\}$.

It then can be shown that

$$P(A(i_1, i_2, \dots, i_p) \rightarrow A(j_1, j_2, \dots, j_q)) = \sum_{0 \leq v \leq p} \left\{ \sum_{(r_1, \dots, r_{q-p+v}) \in S(q-p, v)} m(A(r_1, \dots, r_{q-p+v})) \right\}.$$

For a proof see Kleyle and de Korvin. These general expressions are rather complicated, so in order to get an intuitive feel as to how these transition probabilities are computed, we consider a simple case in which $n = 3$ and there are only 7 focal elements:

$$A_i, i = 1, 2, 3; \quad A_{ij} = A_i \wedge A_j, \quad 1 \leq i < j \leq 3; \\ A_{123} = A_1 \wedge A_2 \wedge A_3$$

where A_1 represents a large enemy force, A_2 represents artillery vehicles, and A_3 represents weapons.

Then using (5) directly we can show that

$$P(A_1 \rightarrow A_1) = m(A_1); \\ P(A_1 \rightarrow A_j) = 0, j = 2, 3 \\ P(A_1 \rightarrow A_{1j}) = m(A_j) + m(A_{1j}), j = 2, 3; \\ P(A_1 \rightarrow A_{23}) = 0 \\ P(A_1 \rightarrow A_{123}) = m(A_{23}) + m(A_{123})$$

Clearly,

$$\sum_{1 \leq j \leq 3} P(A_1 \rightarrow A_j) + \sum_{1 \leq i < j \leq 3} P(A_1 \rightarrow A_{ij}) + P(A_1 \rightarrow A_{123}) = 1$$

Similar transition probabilities are obtained from states A_2 and A_3 to the other states.

REMARK: The transition probability from any first order state into itself is the positive mass associated with that state, but the transition probability from a first order state into any other first order state is zero. This means the probability of staying with the assessment that the object is a large enemy force is positive, but the probability of changing the assessment from a large enemy force to either artillery vehicles or weapons is zero. The transition probability from a first order state into second order state having one of its indices equal to that of the first order state is positive, but the transition probability from a first order state into a second order state not sharing an index with the first order state is zero. That is the probability of changing the decision from a large enemy force to a large enemy force and artillery vehicle or a large enemy force to a large enemy force and weapons is positive, but the probability of changing the decision on a large enemy force to artillery vehicles and weapons is zero. The transition probability from a first order state into the highest order state (third order in this particular situation) is always positive, i.e. the probability of changing the evaluation from a large enemy force to a large enemy force, artillery vehicles and weapons is positive.

We now compute the transition probabilities for second order state A_{12} .

$$P(A_{12} \rightarrow A_i) = 0, \text{ for } i = 1, 2, 3 \\ P(A_{12} \rightarrow A_{12}) = m(A_1) + m(A_2) + m(A_{12}) \\ P(A_{12} \rightarrow A_{ij}) = 0, \text{ for } i \neq 1 \text{ or } j \neq 2 \\ P(A_{12} \rightarrow A_{123}) = m(A_3) + m(A_{13}) + m(A_{23}) + m(A_{123})$$

Similar transition probabilities are obtained for the other two second order states.

REMARK: The transition probability from a second order state to a first order state is zero, and this is true

in general. That is, with the above hierarchical structure on the focal elements, the transition probabilities from a higher to a lower order state is always zero. The transition probability from a second order state into itself is positive, but the transition probability into any other second order state is zero. This pattern is also true in general. The transition probability from a second order state into the highest order state (third order in this situation) is always positive. This means the probability of changing the decision on a large enemy force and artillery vehicles to either a large enemy force, artillery vehicles or weapons is zero, but the probability of reassessing the decision from a large enemy force and artillery vehicles to a large enemy force, artillery vehicles and weapons is positive.

Finally we note that:

$$\begin{aligned} P(A_{123} \rightarrow A_i) &= 0, \text{ for } i = 1, 2, 3 \\ P(A_{12} \rightarrow A_j) &= 0, \text{ for } 1 \leq i < j \leq 3 \\ P(A_{123} \rightarrow A_{123}) &= 1 \end{aligned}$$

REMARK: The transition probability from the highest order state to any lower order state is always zero, and this is also true in general. The transition probability of the highest order state into itself is always 1. That is, the highest order state in this hierarchical structure of intersecting fuzzy states is an absorbing state. In some sense this result is a finite analog of Norton's limit theorem (TH. 4). The interpretation is that once the evaluation is a large enemy force, artillery vehicles and weapons which is the high order state, the probability of changing this assessment to other evaluations is zero.

The above remarks make intuitive sense if the mass function is constructed by a n -fold composition as in eq. (3). Each successive term in the composition represents new information from an independent source. As information accumulates, it is impossible to go back to states associated with less information. (Second order states represent two sources of information, while 3rd order states represent combined information from 3 sources, etc.) The highest order state, representing all accumulated information, is therefore absorbing.

The situation described above represents a hierarchical structure of intersections of fuzzy states in which a D-S type mass function can be used to construct transitional probabilities via eq. (5). This structure is that of the focal elements of an n -fold application of the D-S combination rule, so it is natural that Norton's transition formula, eq. (5), which was derived in the context of the D-S rule works for fuzzy states having this hierarchical

structure. However, the above structure is quite restrictive when states (i.e. focal elements of the mass function) are fuzzy. Furthermore, it is not unique. A much simpler nested structure in which eq. (5) produces valid transition probabilities is given below.

Suppose

$$A_n \subset A_{n-1} \subset \dots \subset A_2 \subset A_1$$

With this nested structure it is easy to show that eq. (5) implies:

$$\begin{aligned} P(A_i \rightarrow A_j) &= 0 \quad \text{for all } i > j \\ P(A_i \rightarrow A_i) &= \sum_{1 \leq k \leq i} m(A_k) \\ P(A_i \rightarrow A_j) &= m(A_j) \quad i < j \end{aligned}$$

Clearly

$$\sum_{1 \leq j \leq n} P(A_i \rightarrow A_j) = 1 \quad \text{for all } i,$$

and A_n is an absorbing state. However, the structure of this model is so restrictive as to make it uninteresting in practice. Other structures on the fuzzy states involving nesting and intersections of the "generating states" can be found for which eq. (5) gives valid transition probabilities, but in general it will not work for fuzzy states (focal elements) as the following simple example illustrates.

Suppose set of aircrafts $X = \{F-14, F-16, A-10, RAH-66\}$ and the fuzzy states are

$$\begin{aligned} A_1 &= .8 / F-14 + .6 / F-16 + .3 / A-10 \\ A_2 &= .2 / F-14 + .5 / F-16 + .7 / RAH-66 \\ A_3 &= .6 / F-16 + .4 / RAH-66 \end{aligned}$$

(7a)

$$m(A_1) = .5 \quad m(A_2) = .4 \quad m(A_3) = .1$$

(7b)

$$\begin{aligned} A_k \wedge A_1 &= A_1 \text{ if and only if } k = 1 \\ \text{so } P(A_1 \rightarrow A_1) &= .5 \end{aligned}$$

But for any

$$\begin{aligned} A_k \wedge A_1 &\neq A_2 \text{ for any } k \\ &\& A_k \wedge A_1 \neq A_3 \text{ for any } k \end{aligned}$$

Thus

$$P(A_1 \rightarrow A_j) = 0 \text{ for } j = 2 \text{ or } j = 3$$

Consequently,

$$\sum_{1 \leq j \leq 3} P(A_1 \rightarrow A_j) = .5 < 1$$

REMARK: A_1 represents the decision from an expert that the object might be .8 F-14, .6 F-16, or .3

RAH-66. And the probability of decision A_1 is .5. Similarly, the probability from an expert's assessment that the object might be .2 F-14, .5 F-16, or .7 RAH-66 is .4; and the probability from an expert's assessment that the object might be .6 F-16, or .4 RAH-66 is .1.

5. The General Situation

As the above example clearly illustrates, eq. (5) will not give meaningful results when the mass function is defined on fuzzy states (i.e. fuzzy focal elements) unless these focal elements have a hierarchical intersection structure, a nested structure or some combination of these structures. There is, in fact, no reason to expect eq. (5) to extend to a situation involving an arbitrary mass function defined on arbitrary fuzzy states, since the Markov chain that Norton works with is related to mass functions obtained by the improper version of the D-S rule given by eq. (2).

Nevertheless, eq. (5) has an intuitive appeal even when the states are fuzzy. Given that we are currently in fuzzy state A_i , the likelihood of going to state A_j when the new information indicates state A_k is the degree to which $A_i \wedge A_k = A_j$. Of course, the new information does not specify state A_k specifically, but gives a mass (i.e. probability) to each state in the chain. A weighted sum over these masses in which the weights indicate the degree to which $A_i \wedge A_k = A_j$ is an intuitively appealing measure of the likelihood of transition from state A_i to state A_j given the new information. What is needed, therefore, to generalize eq. (5) to fuzzy states is a measure of the degree to which two fuzzy sets are equal.

The degree to which two finite fuzzy sets are equal can be defined in various ways. The definition that seems to best fit our purpose is

$$d[A = B] = |A \wedge B| + \text{Max}\{|A|, |B|\} \quad (8)$$

where $|A|$ denotes the scalar cardinality of fuzzy set A . That is,

$$|A| = \sum_{x \in X} A(x)$$

The above definition of degree of equality is based on Ogawa and Fu's (1985) definition of the degree to which fuzzy set A is a subset of fuzzy set B , which is

$$I[A \subset B] = |A \wedge B| + |A| \\ = \sum_{x \in X} \text{Min}\{A(x), B(x)\} + \sum_{x \in X} A(x)$$

(9)

Our definition of degree of fuzzy set equality is simply

$$d[A = B] = \text{Min}\{I[A \subset B], I[B \subset A]\}$$

where $I[A \subset B]$ is given by eq. (9).

Using the definition in (8) we now generalize Norton's eq. (5) to the situation in which an arbitrary mass function is defined on fuzzy states.

$$P(A_i \rightarrow A_j) = \sum_{m(A_k) > 0} d[A_i \wedge A_k = A_j] m(A_k) \\ + \sum_j \sum_{m(A_k) > 0} d[A_i \wedge A_k = A_j] m(A_k)$$

(10)

We can think of the numerator of (10) as the degree of belief that the process moves from fuzzy state A_i to fuzzy state A_j . The denominator is a normalizing factor that converts these transitional beliefs into transitional probabilities.

5.1 Example

Let us now return to the example given above in eqs. (7a) and (7b). Note that

$$A_1 \wedge A_2 = 2 / F - 14 + 5 / F - 16 \\ A_1 \wedge A_3 = .6 / F - 14 + .5 / A - 10 \\ A_2 \wedge A_3 = 5 / F - 16 \\ A_1 \wedge A_2 \wedge A_3 = \emptyset$$

$$\sum_{1 \leq k \leq 3} d[A_1 \wedge A_k = A_1] m(A_k) \\ = d[A_1 = A_1] m(A_1) + d[A_1 \wedge A_2 = A_1] m(A_2) \\ + d[A_1 \wedge A_3 = A_1] m(A_3)$$

NOTE:

$$d[A_1 = A_1] = 1 \\ d[A_1 = A_2] = |A_1 \wedge A_2| + |A_1| = 7 + 17 \\ d[A_1 = A_3] = |A_1 \wedge A_3| + |A_3| = 9 + 17$$

So

$$\sum_{1 \leq k \leq 3} d[A_1 \wedge A_k = A_1] \\ = 5 + (7 + 17) \times .4 + (9 + 17) \times .1 = .7176$$

Similarly we compute:

$$\sum_{1 \leq k \leq 3} d[A_1 \wedge A_k = A_2] = (7+17) \times 5 + (7+14) \times 4 = 4059$$

NOTE:

$$d[A_1 \wedge A_3 = A_2] = |A_1 \wedge A_2 \wedge A_3| + |A_2| = 0.$$

$$\sum_{1 \leq k \leq 3} d[A_1 \wedge A_k = A_3] = (9+17) \times 5 + (9+10) \times 1 = 3457$$

NOTE:

$$d[A_1 \wedge A_3 = A_3] = |A_1 \wedge A_2 \wedge A_3| + |A_3| = 0.$$

Finally, we compute

$$\sum_{1 \leq j \leq 3} \sum_{1 \leq k \leq 3} d[A_1 \wedge A_k = A_j] = 7176 + 4059 + 3457 = 14692$$

Thus,

$$\begin{aligned} P(A_1 \rightarrow A_1) &= .7176 + 14692 = .489 \\ P(A_1 \rightarrow A_2) &= 4059 + 14692 = .276 \\ P(A_1 \rightarrow A_3) &= 3457 + 14692 = .235 \end{aligned}$$

Similarly we compute

$$\begin{aligned} P(A_2 \rightarrow A_1) &= .293 & P(A_2 \rightarrow A_2) &= .554 \\ P(A_2 \rightarrow A_3) &= .153 \\ P(A_3 \rightarrow A_1) &= .255 & P(A_3 \rightarrow A_2) &= .143 \\ P(A_3 \rightarrow A_3) &= .602 \end{aligned}$$

REMARK: For example, $P(A_2 \rightarrow A_1) = .293$ is the probability of changing the decision from A_2 to A_1 is .293, i.e. the probability of changing the decision from the object most likely be RAH-66 to most likely be F-14 is .293.

6. Transition Probabilities From Fuzzy to Crisp States

This will correspond to making a final assessment while our information still yields a fuzzy picture. Thus $P(A \rightarrow j)$ may denote the probability that we go from assessment A (say a large force) to the crisp statement, the size of the force is the number j (where j has some membership in A). We shall implement the technique developed by Smets for this problem. The conversion formula is

$$(11) \quad \begin{aligned} P(A_i \rightarrow j) &= \sum_{1 \leq k \leq n} P(A_i \rightarrow A_k) \{A_k(j) + |A_k|\} \end{aligned}$$

where j denotes a crisp state in state space X . Note that

$$\begin{aligned} \sum_{j \in X} P(A_i \rightarrow j) &= \sum_{1 \leq k \leq n} P(A_i \rightarrow A_k) \sum_{j \in X} \{A_k(j) + |A_k|\} \\ &= \sum_{1 \leq k \leq n} P(A_i \rightarrow A_k) = 1 \end{aligned}$$

To illustrate the use of eq. (11) we return to our example of the previous section. In this example the crisp state space is $X = \{F-14, F-16, A-10, RAH-66\}$, so that

$$\begin{aligned} P(A_1 \rightarrow F-14) &= P(A_1 \rightarrow A_1) \{A_1(F-14) + |A_1|\} \\ &\quad + P(A_1 \rightarrow A_2) \{A_2(F-14) + |A_2|\} \\ &\quad + P(A_1 \rightarrow A_3) \{A_3(F-14) + |A_3|\} \\ &= .489 \times (.8 + 1.7) + .276 \times (.2 + 1.4) = .270 \end{aligned}$$

Recall that $A_3(F-14) = 0$ since state F-14 is not in the support of A_3 .

In a similar manner we compute:

$$\begin{aligned} P(A_1 \rightarrow F-16) &= .489 \times (.6 + 1.7) + .276 \times (.5 + 1.4) \\ &\quad + .235 \times (.6 + 1.0) = .412 \\ P(A_1 \rightarrow A-10) &= .489 \times (.3 + 1.7) + .235 \times (.4 + 1.0) \\ &= .180 \\ P(A_1 \rightarrow RAH-66) &= .276 \times (.7 + 1.4) = .138 \end{aligned}$$

Similarly:

$$\begin{aligned} P(A_2 \rightarrow F-14) &= .217 \\ P(A_2 \rightarrow F-16) &= .393 \\ P(A_2 \rightarrow A-10) &= .113 \\ P(A_2 \rightarrow RAH-66) &= .277 \\ P(A_3 \rightarrow F-14) &= .140 \\ P(A_3 \rightarrow F-16) &= .502 \\ P(A_3 \rightarrow A-10) &= .286 \\ P(A_3 \rightarrow RAH-66) &= .072 \end{aligned}$$

REMARK: For example, $P(A_2 \rightarrow F-14) = .217$ is the probability of changing the decision from A_2 to F-14 is .217, i.e. the probability of changing the decision from the object most likely be RAH-66 to definitely be F-14 is .217.

7. Summary

We have used a modification of a procedure defined by Norton (1988) to be able to estimate the probability of staying with our original assessment or

changing it by incorporating information refinement. We also have estimated the probability of going from a fuzzy assessment to a definite crisp evaluation. Our procedure yields naturally a "limiting set" which represents the limit of attainable assessment using the sources of information available. (If that set is empty, the information is found to be inconsistent.)

8. Acknowledgments

This research is partially supported by an ARO grant number DAAH-0495-1-0250 and an NSF grant number CDA-9522157.

9. References

- de Korvin, A., Hashemi, S., Quirchnayr, G. and Hord, S., Evaluating policies based on their long term average cost. (Submitted)
- Dempster, A. P., (1967), Upper and lower probabilities induced by a multivalued mapping. *Annals of Mathematical Statistics*, 38, 325-339.
- Dubois, D. and Prade, H. (1980), *Fuzzy Sets and Systems: Theory and Applications*, Academic Press, New York.
- Handley, J., Jaenisch, H., and Carroll, M. (1995), Identification of the behavior transition point in data using a robust fractal method. *Proceedings of the Southeastern Simulation Conference, SESC '95*, October 1995, Orlando Florida, 57-66.
- Hille, D., Hieb, M., Tecuci, G., and Pullen, J. (1995), Abstracting terrain data through Sematic terrain transformations. *Proceedings of the Fifth Conference on Computer Generated Forces and Behavioral Representation*, May 9-11, 1995, Orlando, Florida, 355-365.
- Holmes, W. (1995), Fuzzy systems computational and simulation models for missile system control. *Proceedings of the Southeastern Simulation Conference, SESC '95*, October 1995, Orlando Florida, 37-44.
- Klement, E., Schwyhla, W. and Lowen, R., (1981), Fuzzy probability measures. *Fuzzy Sets and Systems*, 5, 21 -30.
- Kleyle, R. and de Korvin, A., Policy selection based on a Markov model with fuzzy transition probabilities. (Submitted).
- Kleyle, R. and de Korvin, A., Transition probabilities for Markov chain having fuzzy states. *Journal of Stochastic Analysis and Applications*, to appear.
- Klir, G. and Folger, T., (1988), *Fuzzy Sets, Information and Uncertainty*, Prentice Hall, Englewood Cliffs, NJ.
- Norton, I. (1988), Limit theorems for Dempster's rule of combination. *Theory and Decisions*, 25, 287-313.

- Ogawa, H. and Fu, K.S., (1985), An inexact inference for damage assessment of existing structures. *Int. Journal of Man-Machine Studies*, 22, 295-306.
- Paisecki, K., (1985), Probability of fuzzy events defined as denumerable additivity measure. *Fuzzy Sets and Systems*, 17, 271-284.
- Pandari, P. and Schaper G., Terrain reasoning by intelligent player. *Proceedings of the Fifth Conference on Computer Generated Forces and Behavioral Representation*, May 9-11, 1995, Orlando, Florida, 367-373.
- Shafer, G., (1976), *A Mathematical Theory of Evidence*, Princeton University Press Princeton, NJ.
- Smets, P., (1982), Probability of a fuzzy event: an axiomatic approach. *Fuzzy Sets and Systems*, 7, 153-164.
- Smets, P., (1990), Belief functions versus probability functions. *Uncertainty in Artificial Intelligence*, 5, 1-8.
- Yin, Y. and Chung, K. (1995), A heuristic search method for path-constrained, Markov-moving target. *Proceedings of the Southeastern Simulation Conference, SESC '95*, October 1995, Orlando Florida, 251-257.
- Zadeh, L., (1965), Fuzzy Sets. *Information and Control*, 8, 338-353.
- Zadeh, L., (1968), Probability measures of fuzzy events. *Journal of Mathematical Analysis and Applications*, 23, 421-427.
- Zadeh, L., (1979), Fuzzy sets and information granularity. *Advances in Fuzzy Set Theory and Applications*, 3- 18.

10. Biographies

Richard A. Aló is a professor and chair of Computer and Mathematical Sciences Department at the University of Houston-Downtown. His research interests are in the areas of automated reasoning and decision making, computability/complexity, numerical analysis, and functional analysis.

Mohsen Beheshti is an assistant professor of Computer and Mathematical Sciences Department at the University of Houston-Downtown. His research interests are in the areas of object-oriented database, logical optimization using fuzzy logic, and concurrency control.

Andre de Korvin is a professor of Computer and Mathematical Sciences Department at the University of Houston-Downtown. His research interests are in the areas of fuzzy logic, rough sets neural nets, and probability theory.

Chenyi Hu is an assistant professor of Computer and Mathematical Sciences Department at the University of Houston-Downtown. His research interests are in the areas of reliable scientific computation and parallel processing.

Ongard Sirisaengtaksin is an associate professor of Computer and Mathematical Sciences Department at the University of Houston-Downtown. His research interests are in the areas of neural networks, fuzzy logic, and intelligent control systems.



Simulating a Battlefield Maneuver Using Reaction Diffusion Equations

MaryAnne Fields
Army Research Laboratory
Weapons Technology Directorate
Attn: AMSRL-WT-WE
Aberdeen Proving Ground, Maryland

1. Abstract

In this paper we present a battlefield simulation model which uses a set of partial differential equations, called reaction diffusion equations, to model the behavior of each army on the battlefield. The model is used to simulate the attack of a stationary red force by a blue force. In this simulation, the blue army is divided into three smaller forces. Two of the blue forces envelop the red army while the third force conducts a diversionary frontal attack. We use the reaction diffusion equation based model to study the effects of variation of model parameters on the overall battle dynamics.

2. Introduction

The purpose of this paper is to present an aggregate level computer generated force model which uses a family of parabolic partial differential equations called reaction diffusion equations (RDEs) to model the movement and interactions of the troops on the battlefield. The primary assumption of an RDE based battlefield simulation is that troops tend to move and act as groups rather than individuals. The behavior of these groups is described by a system of RDEs similar to those used in other fields to describe the movement, spread, and interaction of biological or chemical species.

We are developing RDE based battlefield simulations to use a parametric analysis tools in the study of weapon concepts for the battlefield. Preliminary work on RDE based battlefield

simulation is described in the papers of Fields (1993), Azmy (1991), Protopopescu et al. (1989), and Santoro et. al. (1989).

In this paper, we describe the RDE battlefield simulation model in general. We apply this model to a scenario in which a stationary red force is attacked by a blue force. In the simulation, the blue force is subdivided into two enveloping forces and a third force that conducts a diversionary frontal assault.

3. An Overview of the RDE Battlefield Simulation Model

Let B be a region of \mathbb{R}^2 representing the battlefield and A_i , $1 \leq i \leq n$, be the armies involved in the battle. In this paper, the term army refers to a homogeneous force of at least battalion size. A general RDE describing the movement and interaction of the army A_i is given by:

$$a_{i,t} = D_{i,x} a_{i,xx} + D_{i,y} a_{i,yy} + V_{i,x} a_{i,x} + V_{i,y} a_{i,y} - I_i \quad (1)$$

in which $a_i(x,y,t)$ measures the strength of the army A_i at the point (x,y) at time t . Strength may be either a measure of the number of troops in a given area of the battlefield or a measure of the combat power of those troops as described by Dupuy (1987). In this paper, strength measures the number of units (or troops) at each point (x,y) .

Movement of the army A_i , as modeled by an RDE has two components: diffusion and convection. The more important of the two, convection, determines the primary direction of movement for the army. In many cases, it also determines the speed of that movement, are least to a first order approximation. In Equation 1, the coefficient functions $V_{i,x}$ and $V_{i,y}$

control the convective movement of the army in the x and y directions, respectively. These coefficients are functions of several factors including the operational orders of the army, the terrain, and the distribution of other armies on battlefield. The coefficient functions V_{i_x} and V_{i_y} will be discussed in greater detail in the next section.

The second component of movement in the RDE, diffusion, models the natural tendency of troops to move randomly. Diffusion of army A_i is controlled by the coefficient functions D_{i_x} and D_{i_y} . Large diffusion coefficients model scenarios in which troops spread out while traveling. Variations in D_{i_x} and D_{i_y} as functions of the terrain, can be used to simulate battlefields containing marshes or other regions in which the troops have a natural tendency to spread out. In model presented in this paper, the diffusion coefficients are set to a small constant value simulating tightly controlled movement.

On the battlefield, troops are generally lost through attrition and gained through reinforcement. The last term on the right hand side of Equation 1, I_i models the net gain or loss of army A_i as a function of time, space and the other armies on the battlefield.

4. Simulating an Envelopment Maneuver

In this paper, we simulate an envelopment maneuver involving three blue armies (armies A_1 , A_2 , and A_3) and two red armies (armies A_4 and A_5). In this simulation, we assume that the red army has had time to prepare offensive positions and that it is "dug-in", so these forces do not move. The red army is subdivided into two forces to simulate a front line force which have direct contact with the enemy and a artillery force which does not have direct contact with the enemy. Army A_3 has direct line-of-sight with the enemy forces. Army A_2 is a non-line-of-sight force which responds to calls for fire from army A_4 . The blue army is divided into three armies, each with a different mission. Armies A_1 and A_2 envelop the red force from different

directions while army A_3 conducts a diversionary frontal attack.

The RDE system which models the envelopment maneuver is

$$\begin{aligned} a_{1_t} &= Da_{1_{xx}} + Da_{1_{yy}} + F_x a_{1_x} + F_y a_{1_y} - I_1(a_4, a_5) \\ a_{2_t} &= Da_{2_{xx}} + Da_{2_{yy}} + E_{c_x} a_{2_x} + E_{c_y} a_{2_y} - I_2(a_4, a_5) \\ a_{3_t} &= Da_{3_{xx}} + Da_{3_{yy}} + E_{cc_x} a_{3_x} + E_{cc_y} a_{3_y} - I_3(a_4, a_5) \\ a_{4_t} &= -I_4(a_1, a_2, a_3) \\ a_{5_t} &= -I_5(a_1, a_2, a_3). \end{aligned} \quad (2)$$

The diffusion coefficients for the attacking armies are set to a small positive number, simulating tight formations. The name of the convective coefficients in the first three equations indicates the type of maneuver used by each army. The first army is the frontal attack force, the second army envelops the red army in a clockwise direction, the third army envelops the red army in a counterclockwise direction. The two red armies are "dug-in" so that they cannot move. However, they can lose troops through attrition.

The net loss for an attacking army A_i ($i = 1, 2$, or 3) at the point (x, y) is given by the

$$I_i(x, y) = a_i(x, y) \iint_{\mathcal{R}} f_{i_k}(a_4) + f_{i_5}(a_4, a_5) d\mathcal{R} \quad (3)$$

where \mathcal{R} is a region of the battlefield centered at (x, y) . In this model, \mathcal{R} is a circular region whose radius is determined by the maximum range of the weapons involved. The functions f_{i_k} , ($k = 4$ or 5) determines the effectiveness of the army A_k against army A_i as a function of range.

It is possible to simulate communication and cooperation among the red forces. In particular, we can simulate a "call-for-fire" by having the rear forces respond to what the front line forces see.

Our previous work with the RDE battlefield simulation model focused on developing the convective coefficient functions to model various types of movement on the battlefield. We developed models of troop movement on realistic terrain. We use work in this paper but, we will concentrate on three types of movements used in offensive operations - movement to a specific point on the

battlefield, a frontal attack and an enveloping attack. A more complete discussion of terrain related aspects of the coefficient functions V_{ix} and V_{iy} is given in Fields (1993) and Fields (1995) .

The terrain used throughout this paper is an artificial surface referred to as the variable resolution terrain (VRT) model. It is continuously differential surface constructed by summing several hill functions, (similar to bivariate normal functions) of various sizes. We use it in conjunction with the RDE battlefield simulation model because it is continuously differentiable and because we can design terrain surfaces to test specific features of the RDE model. VRT can also be used to fit actual terrain data sets. By its nature, VRT is a complex surface with non uniformity at any desired level of detail. Consequently, it can supplement an actual terrain data set with realistic micro terrain features. For applications such as infantry combat simulations, the micro terrain features add necessary realism to standard terrain data sets in which elevation posts are often 100 meters apart. For more information on the VRT model, the reader is referred to the report by Wald and Patterson (1992) and the paper of Wald (1994) .

4.1 Movement to a Rendezvous Point

In this section, we model a simple operational order which directs the troops to move to the military objective such as rendezvous point. We want the simulation to be as flexible as possible so we do not want to stipulate an initial position for the troops. Instead, we develop coefficients so that troops can reach the rendezvous point from almost any position on the battlefield. It is helpful to visualize V_{ix} and V_{iy} as components of a velocity vector, v , and to discuss the velocity vector field generated by v . This vector field is similar to a magnetic or electric field and governs the movement of troops at any position on the battlefield.

Figure 1 shows a velocity vector field that directs the troops to the black circle in the upper right section of

the figure. The vector field is oriented so that the variable x changes in the horizontal direction and the variable y changes in the vertical direction. The figure also shows ten equally spaced contours. The height of some points of the battlefield have been labeled to aid the reader. The length and width of the battlefield are both 10 kilometers.

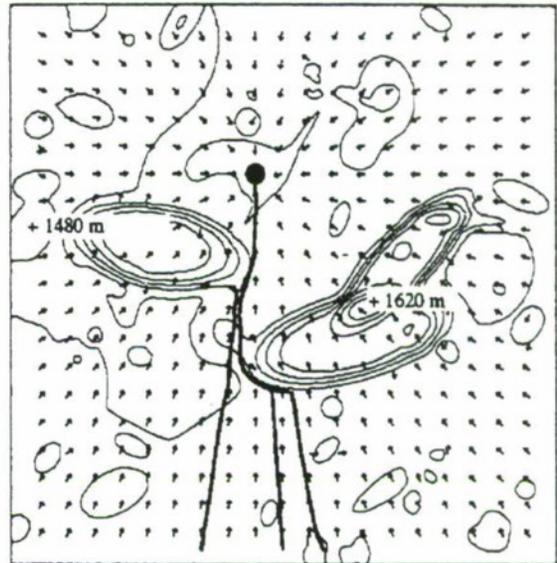


Figure 1. A Velocity Vector Field.

The direction of each arrow on the battlefield is a function of terrain characteristics such as steepness and elevation and the operation orders of the army A_i . The vector function is given by

$$v = w_d d + w_t (\alpha_c t_c^+ + \alpha_{cc} t_{cc}^+) + w_s s \quad (4)$$

The vector d is determined by the operational orders for the army and by the army's ability to respond to obstacles on the battlefield. In this example, the orders direct the troops to move to the rendezvous point indicated by the black circle shown in Figure 1. Suppose the rendezvous point is at (x_0, y_0) , then at any point (x, y) on the battlefield ,

$$d(x, y) = \frac{v}{\epsilon + \sqrt{(x - x_0)^2 + (y - y_0)^2}} \begin{pmatrix} x - x_0 \\ y - y_0 \end{pmatrix}. \quad (5)$$

If ϵ is small, then speed of the troops will be close to v nearly everywhere on the battlefield. This vector uses only local information to determine the direction of troop movement. By incorporating

"regional" information, the troops can respond to obstacles in the distance.

The vectors t_c^\perp and t_∞^\perp are functions of terrain of the battlefield. The gradient of the terrain, which gives the direction of steepest ascent from the point (x,y) , is $t(x,y) = (T_x, T_y)$ where $T(x,y)$ is the terrain surface. The vectors t_c^\perp and t_∞^\perp are orthogonal to t . By following t_c^\perp or t_∞^\perp , troops travel along a contour of the surface in a clockwise or counterclockwise direction, respectively.

The vector s is derived from the steepness of the terrain. Let the steepness of a point (x,y) be defined as $S(x,y) = \frac{1}{2}(T_x^2 + T_y^2)$, which is one half the square of the length of the terrain gradient vector at the point (x,y) , then the steepness gradient is defined as

$$s(x,y) = \begin{pmatrix} T_x T_{xx} + T_y T_{yx} \\ T_x T_{xy} + T_y T_{yy} \end{pmatrix}. \quad (6)$$

The weights w_d , w_t and w_s are functions of the steepness of the terrain. Each point (x,y) on the battlefield is categorized as easy, moderate, difficult or impassable as a function of the steepness of the terrain. For each of these category, a different vector dominates the movement vector field. For instance, in regions of the battlefield categorized as "easy", the vector d dominates the vector field. The weights are sigmoidal or bell-shaped functions which turn "on" for certain steepness values and "off" for other values.

The weights are also functions of the movement characteristics of the simulated armies. By varying the weighting functions, it is possible to increase or decrease the army's sensitivity to the terrain. In this paper, we shall use weights which allow the troops to move freely throughout most areas of the battlefield. Near the two large hills in the center of the battlefield, troops must adjust their movements to avoid the steep regions.

4.2 Frontal Attack

In this section, we design a vector field to simulate the frontal attack on a stationary force. Suppose in this simulation the army A_i is the attacker and the army A_k is the defender. In designing this field, we want to consider the following elements. First, this is a planned attack so the army A_i has orders to attack the army A_k . The frontal attack vector field depends on vectors similar to those in Equation 3 to directs troops to a military objective within the terrain occupied by the army A_k while avoiding obstacles in the terrain. In this vector field, we also need to control the attack so that it is realistic. The frontal attack vector field is a function the position and strength of both armies.

In the RDE battlefield simulation model, the armies are represented by continuous, non-negative, distribution functions a_k and a_i , respectively. Let us define a local force ratio function, r_i , for the attacking army A_i as the ratio of the strength of A_i to the strength of A_k within a neighborhood \mathcal{N} around the point (x,y)

$$r_i(x,y) = w_k \frac{\iint_{\mathcal{N}} a_i d\mathcal{N}}{\iint_{\mathcal{N}} a_k d\mathcal{N} + \delta}. \quad (7)$$

The term δ is a small positive constant therefore r_i is defined at all points on the battlefield. If w_k is a constant, the force ratio is independent of the actual number of combatants within the neighborhood \mathcal{N} . For realism, the force ratio is only important if there is enough of the enemy force in the neighborhood \mathcal{N} to attack. Let

$$w_k(x,y) = \frac{e^{\beta(\iint_{\mathcal{N}} a_k d\mathcal{N} - \alpha_i)}}{1 + e^{\beta(\iint_{\mathcal{N}} a_k d\mathcal{N} - \alpha_i)}} \quad (8)$$

Then for values of $\iint_{\mathcal{N}} a_k d\mathcal{N} < \alpha_i$, the force ratio is near zero; for values of $\iint_{\mathcal{N}} a_k d\mathcal{N} > \alpha_i$ the force ratio reflects the relative strength the armies.

The vector field for the frontal attack is given by

$$v = w_d d + w_t (\alpha_c t_c^\perp + \alpha_\infty t_\infty^\perp) + w_s s + f_1(r_i) a_k + f_2(r_i) a_{k_c}^\perp + f_3(r_i) a_{k_\infty}^\perp. \quad (9)$$

The first three vectors on the right hand side are similar to those used in Equation 3. The gradient vector a_k points in the direction of increasing strength for the army A_k ; $a_{k_c}^+$ and $a_{k_{cc}}^+$ are orthogonal to the gradient of A_k . The functions f_1 , f_2 and f_3 , which are functions of the steepness of the terrain as well as functions of the force ratio, determine the weight of the attack vectors in the overall vector field. There may be points on the battlefield at which the force ratio may dictate an attack at a given point on the battlefield, but terrain features are too severe to allow the attack. By adjusting the weights on the attack vectors, the army responds to the terrain rather than to the opposing army at those points.

The vector field generated by a frontal attack scenario changes continually as forces move on the battlefield. When the attacking force is far away from the red force, the changes in the field are slight. However, as the blue force begins to make contact with the red force, the local force ratio starts to decrease, changing the vector field. As more of the attacking force begins to make contact with the enemy, units of the attacking force adjust their paths to maintain a force ratio. Figure 2 shows four paths generated by an attack scenario. The location of the red armies are indicated on the contour map, the blue army starts its advance at the bottom center of the figure. Each of the paths are generated at a different time in the scenario. Path #1 shows the initial contact with the red army, paths #2-#4 show later contacts with the enemy.

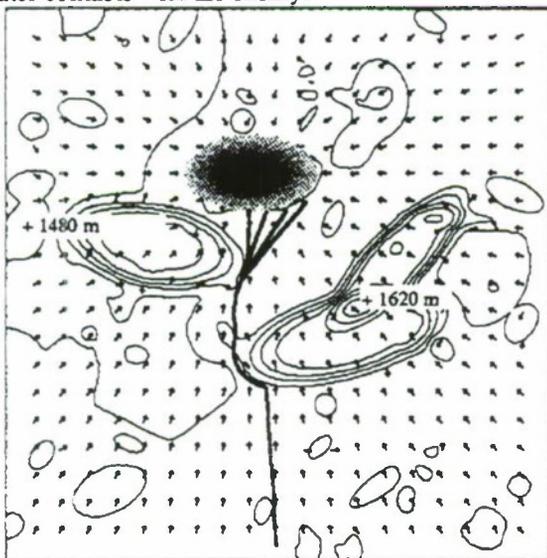


Figure 2. A Frontal Attack

4.3 Envelopment

Constructing a vector field to describe an envelopment of a stationary army is very similar to constructing a vector field to describe a frontal attack of a stationary army. In an envelopment, the attacking army bypasses the front line and attempts to attack the army from the rear. In our example there are two envelopment - one in a clockwise direction, the other in a counterclockwise direction. In functional form, the vector field describing the enveloping attack is the same as the vector field describing the frontal attack given in Equation 6. The functions f_1 , f_2 and f_3 are different.

An example of an envelopment is shown in Figure 3. The attacking army envelops the stationary army in a clockwise direction. Since a hill is blocking the way, the army also proceeds clockwise around the hill.

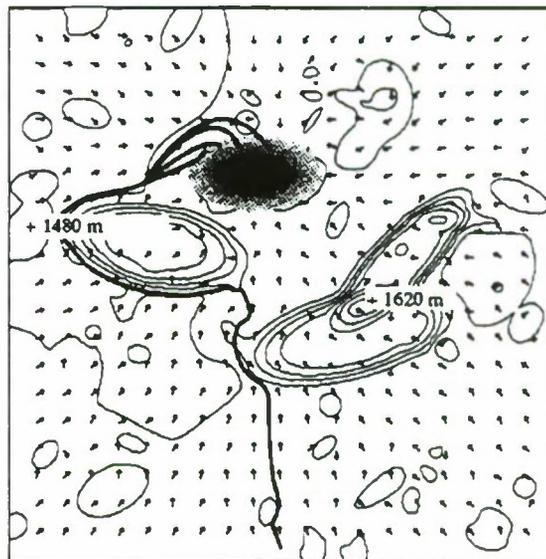


Figure 3. An Envelopment Maneuver.

4.4 Simulating the Attack

By combining using the vector fields generated in the previous sections as the convective coefficient functions for the equations controlling the attacking army, it is possible to simulate an envelopment

scenario. The general attack direction for each blue army is given in Figure 4.

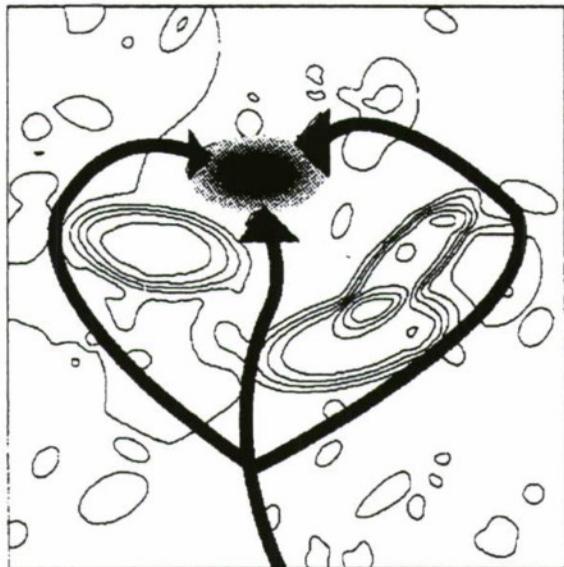


Figure 4. An Envelopment Scenario.

At the present time, the simulation is implemented on a multiprocessor computer. The equations controlling each of the five armies are solved by different group of processors. As the complexity of the vector fields increase, the number of processors needed to solve the equations efficiently increases.

5. Movement as a Function of Terrain Resolution

In this section, we illustrate a possible use for the RDE model as a tool for parametric analysis of battle parameters. In this example, we examine the sensitivity of our troop movement model to terrain resolution. Troop movement, as modeled by the RDE battlefield simulation, is sensitive the steepness of the terrain. Steepness for a digital terrain data set is a function of the resolution of the data set. Even if the steepness is known exactly at the elevation

posts of the data set, it must be interpolated in between those posts. In general, higher resolution data sets have more variation in steepness than lower resolution data sets.

Figures 4 and 5 show the same VRT surface sampled at two different resolutions. The VRT surface represents a 10km x 10km battlefield. The figures show ten equally spaced contours and the highest and lowest elevations on the battlefield. In Figure 4, elevation posts were 10 meters apart. In Figure 5, elevation posts are 100 m apart. Although the figures show the same large features, many of the smaller features shown in Figure 4 are missing or distorted in Figure 5.

The paths shown in Figures 5 and 6 illustrate the effect of terrain resolution on the movement of the simulated forces. The paths in Figure 5 are longer than the corresponding paths in Figure 6, indicating that our movement model is sensitive to the resolution of the battlefield.

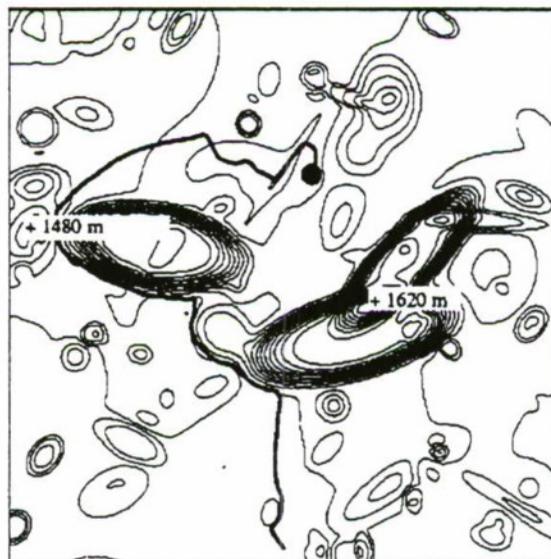


Figure 4. Movement on a 10m resolution battlefield.

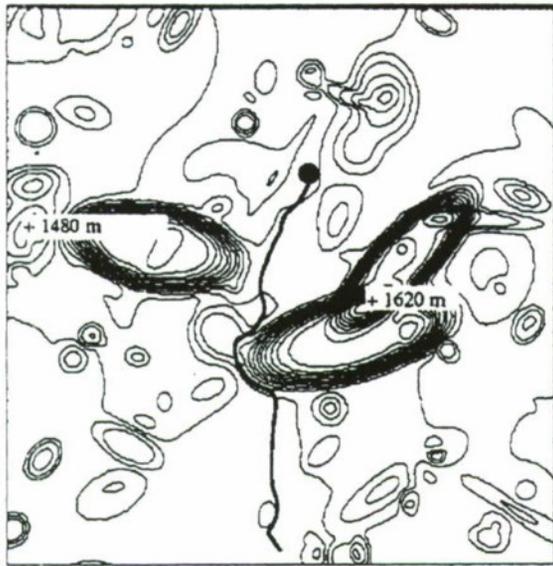


Figure 5. Movement on a 100m resolution battlefield.

6. Conclusion

In this paper, we have developed a battlefield simulation model based on RDEs. By incorporating terrain and battlefield intelligence information in the coefficient functions, we have shown that the RDE model is flexible enough to simulate a complex attack scenario. In the example scenario presented, we simulate an enveloping attack of a stationary red force by a blue force. The blue army is subdivided into two enveloping forces and a frontal attack force. The red army is divided into forces which have direct line-of-sight to enemy forces and non line-of-sight forces.

RDE based battlefield simulation offer a method to mathematically study the dynamics of a battlefield simulation in a controlled setting. Using methods for the qualitative analysis of differential equations, we can study the sensitivity of the model to its parameters. At the present time, this will help us build a more realistic simulation tool. In the future, after we have matched our parameters to real data, we can use the RDE model to investigate weapon system concepts for the battlefield, particularly at the early stages of the development process.

7. References

- Azmy, Y.Y. (1991). *DCOR: A Deterministic Combat Model Code.*, ORNL/TM-11690, Oak Ridge National Laboratory, Oak Ridge, TN.
- Dupuy, T.N. (1987). *Understanding War: History and a Theory of Combat*, Paragon House.
- Fields, M.A. (1993). *Modeling Large Scale Troop Movement Using Reaction Diffusion Equations*, ARL-TR-200, U.S. Army Research Laboratory, Aberdeen Proving Ground, MD.
- Fields, M.A. (1995). *Preliminary Applications of the Variable Resolution Terrain Model to a Troop Movement Model*, ARL-TR-842. U.S. Army Research Laboratory, Aberdeen Proving Ground, MD.
- Protopopescu, V., R.T. Santoro and J. Dockery (1989). *Combat Modeling with Partial Differential Equations.*, ORNL/TM-10636, Oak Ridge National Laboratory, Oak Ridge, TN.
- Santoro, R.T. , P. Rusu and J.M. Barnes, (1989). *A Mathematical Description of Offensive Combat Maneuvers.*, ORNL/TM-10000, Oak Ridge National Laboratory, Oak Ridge, TN.
- Wald, J.K. and C.J. Patterson (1992). *A Variable Resolution Terrain Model for Combat Simulation*, BRL-TR-3374, U.S. Army Ballistics Research Laboratory, Aberdeen Proving Ground, MD.
- Wald, J.K. (1994). *Solving the "Inverse" Problem in Terrain Modeling*, ARL-TR-605. U.S. Army Research Laboratory, Aberdeen Proving Ground, MD.

8. Biography

MaryAnne Fields is a mathematician in the Weapons Concepts Division of the Army Research Laboratory. Dr. Fields has a Ph.D in applied mathematics. Her research interests are in the areas of applied differential equations, parallel processing and computer generated forces.



Flexible Teamwork for Intelligent Simulated Pilots

Milind Tambe

Information Sciences Institute and Computer Science Department
University of Southern California
4676 Admiralty Way, Marina del Rey, CA 90292
tambe@isi.edu
<http://www.isi.edu/soar/tambe>

Abstract

Flexible teamwork among synthetic agents is a critical capability in advanced distributed combat simulations. Such teamwork is more than a simple union of agents' simultaneous execution of individual plans, even if such execution pre-coordinated. Indeed, uncertainties in complex, dynamic combat simulations often obstruct pre-planned coordination, with a resultant breakdown in teamwork. The central hypothesis in this paper is that for durable teamwork, agents should be provided explicit *team plans* and an underlying model of teamwork that explicitly outlines their commitments and responsibilities as participants in team activities. Such a model enables team members to flexibly reason about coordination activities. The underlying model we have provided is based on the *joint intentions* framework; although we present some key modifications to reflect the practical constraints in (some) real-world domains.

This framework has been implemented in the context of a company of synthetic helicopter pilot agents; some empirical results are presented.

1 Introduction

The Soar-IFOR (Tambe *et al.* 1995; Tambe, Schwamb, & Rosenbloom 1995) project has been developing intelligent automated pilots for participation in advanced distributed combat simulations. Since July 1994, we have been developing pilot agents for synthetic rotary-wing aircraft (RWA), specifically, synthetic AH-64 Apache attack helicopters. In our previous work, reported in the proceedings of this conference last year (Tambe, Schwamb, & Rosenbloom 1995), we focused pilot agents for individual attack helicopters. This paper goes beyond by focusing on a company of attack helicopters, which may involve up to eight pilot agents.

The key issue addressed in this paper is enabling flexible teamwork within such a team of synthetic agents.¹ Such teamwork is not merely a union of simultaneous, coordinated individual activities (Grosz & Sidner 1990; Cohen & Levesque 1991). For instance, ordinary auto-

¹In this paper, the word "team" should be interpreted in its general sense, as referring to units such as a company, a platoon, a section, etc.

mobile traffic is not considered teamwork, despite the simultaneous activity, coordinated by traffic signs (Cohen & Levesque 1991). On the contrary, driving in a convoy, even if sometimes uncoordinated is considered teamwork. Indeed, our commonsense notion of teamwork involves more than simple coordination, e.g., the American Heritage Dictionary defines it as *cooperative effort by the members of a team to achieve a common goal*.

Yet, to sustain such cooperation in complex, dynamic combat simulations, agents must be flexible in their coordination and communication actions, or else risk a breakdown in teamwork. To achieve such flexibility we apply one key lesson from the arena of knowledge-based systems — an agent must be provided explicit "deep" or causal models of its domains of operation (Davis 1982). The key here is to recognize that when an agent participates in a team activity, teamwork is itself one of the domains, and hence the agent must be provided an explicit model of teamwork. Unfortunately, among implemented agents in advanced distributed combat simulations, team activities and the underlying model of teamwork are often not represented explicitly (Jennings 1994; 1995). Instead, individual agents are often provided individual plans to achieve individual goals, with detailed precomputed plans for coordination and communication. However, in real-world combat simulations unanticipated events often disrupt preplanned coordination, jeopardizing the team's joint effort (the next section provides detailed examples).

The recent formal theories of collaborative action have begun to provide the required models for flexible reasoning about team activities (Cohen & Levesque 1991; Grosz & Sidner 1990; Kinny *et al.* 1992; Jennings 1995); although few multi-agent implementations have built up on them (Jennings 1995). In contrast, this paper describes an implemented, real-world multi-agent system that builds upon one such model. Our central hypothesis is that for effective teamwork in complex, dynamic domains, individual team members should be provided *team goals/plans*, that explicitly express a team's joint activities — although these may hierarchically expand out into goals/plans for an individual's role in the team. To execute such team plans, team members must be provided an explicit model

of teamwork — their commitments and responsibilities as team members — so they can flexibly reason about coordination and communication. In our work, this model is the formal *joint intentions* framework(Cohen & Levesque 1991), which we have modified in key ways to accommodate the constraints that appear typical in (some) real-world dynamic domains.

Before describing team plans in detail, we first concretely motivate their need by describing our initial experiences in designing a company of helicopter pilot agents. While we focus on a helicopter company, the lessons learned here appear general enough to be applicable to other agent-teams in advanced distributed combat simulations. All our implementations are based on the Soar architecture(Newell 1990; Rosenbloom *et al.* 1991). We assume some familiarity with Soar’s problem-solving model, which involves applying an operator hierarchy to states to reach a desired state.

2 Initial Experiences

Figure 1 shows a typical attack mission for a company of attack helicopters. The company may fly 25-50 kilometers at varying altitudes, to halt at a holding point. One or two *scout* helicopters in the company fly forward to check the battle position, i.e., the location from where the company will attack enemy forces. Once the battle position is scouted, other members of the company move forward, each hovering in its own designated subarea of the battle position. Here, an individual pilot agent hides/masks its helicopter. To attack, the pilot has his helicopter “pop-up” (rise high), to shoot missiles at enemy targets. The helicopter then quickly masks and moves as protection against return fire, before popping up again. When the mission completes, the helicopters regroup and return to base.

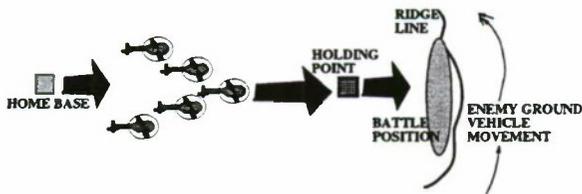


Figure 1: A company of helicopters in simulated combat. The ridge line is ideal for masking.

In our first implementation of the helicopter company, each pilot agent was provided an operator hierarchy to execute its mission(Tambe, Schwamb, & Rosenbloom 1995). Figure 2 illustrates a portion of this operator hierarchy (at any one time, only one path in this hierarchy from the root to a leaf node is active). Each operator consists of (i) precondition rules, to help select the operator; (ii) application rules to apply the operator once selected (a high-level, non-leaf operator may subgoal); (iii) termination rules, to terminate the operator.

To coordinate among multiple pilot agents we used techniques quite comparable to previous such efforts, including

our own, in the synthetic battlefield domain(Tambe *et al.* 1995; Rajput & Karr 1995; Tidhar, Selvestrel, & Heinze 1995). In particular, each individual was provided specific plans to coordinate with others. For instance, when at the holding point, the scout first executed an operator to fly to the battle position, and then another operator to inform those waiting at the holding point that the battle position is scouted. Similarly, to fly in formation, each agent was assigned a “partner” agent to follow in formation (unless the agent was leading the formation). Eventually, all coordination within a group was accomplished by each agent coordinating with its partner.

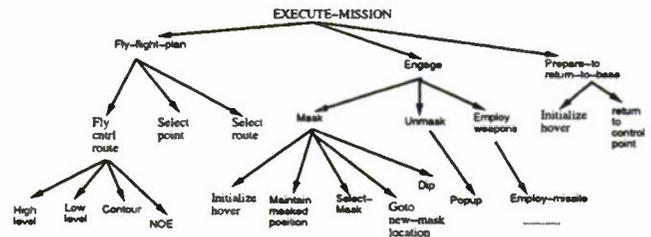


Figure 2: A portion of the operator hierarchy for an individual helicopter pilot agent.

The resulting pilot agents each contained about 1000 rules, and the company was tested in October 1995 in the three-day ED-1 exercise (with upto 400 agents in the synthetic battlefield).² While the helicopter company executed helicopter tactics adequately, the exercise revealed some key problems in teamwork — see Figure 3 for some illustrative examples.

While a programmer could add specialized coordination actions to address the above failures once discovered, anticipating such failures is extremely difficult, particularly as we scale-up to increasingly complex team missions. Instead, the approach pursued in this work is to focus on the root of such teamwork failures — that as with other multi-agent systems, individual team members have been provided fixed coordination plans, which break down when unanticipated events occur. In particular, the team goals and/or team plans are not represented explicitly. Furthermore, an underlying model of teamwork, spelling out team members’s commitments and responsibilities towards others when executing a team activity, is absent. That is why, for instance, an agent ends up abandoning its team members in a risky situation (Item 2, Figure 3). That is also why the company cannot recover when the scout crashes (Item 1, Figure 3) — there is no explicit representation of the company’s team goal at the holding point and the scout’s part in it.

3 Explicit Model of Teamwork

To provide agents with an explicit model of teamwork, we rely on the *joint intentions* framework(Cohen & Levesque

²The ED-1 test for Soar/IFOR helicopter pilot agents was itself a team effort, led by Paul Rosenbloom; Karl Schwamb and the author were the other members involved in the test.

1. Upon reaching the holding area, the company waited, while the scout started flying forward. Unfortunately, the scout unexpectedly crashed into a hillside. Hence, the rest of the company just waited indefinitely at the holding area, waiting to receive a message from the (crashed) scout that the battle position was scouted.
2. Upon recognizing that the mission was completed, one company member (the commander) returned to home base, abandoning others at the battle position. The commander's "partner" agent was unexpectedly shot down, and hence it failed to coordinated with others in its company.
3. While attacking the targets from the battle position, only one member of the company could see the targets. Thus, only one member engaged the targets; the others returned without firing a single shot.
4. Some company members failed to recognize that they had reached a waypoint — the agent leading the formation had reached the waypoint, but those trailing in formation concluded they had not individually done so (despite tolerance ranges in measuring distances).

Figure 3: Some illustrative examples of breakdown in teamwork.

1991; Levesque, Cohen, & Nunes 1990), since currently it is perhaps the most well-understood framework. In this framework, a team Θ jointly intends a team action if team members are jointly committed to completing that team action, while mutually believing that they were doing it. A joint commitment in turn is defined as a joint persistent goal (JPG). A JPG to achieve p , where p stands for completion of a team action, is denoted $JPG(\Theta, p)$. $JPG(\Theta, p)$ holds iff three conditions are satisfied³:

1. All teammates mutually believe that p is currently false.
2. All teammates mutually know that they want p to be eventually true.
3. All teammates mutually believe that until p is mutually known to be achieved, unachievable or irrelevant, they mutually believe that they each hold p as a weak goal (WG). $WG(\mu, p, \Theta)$, where μ is a team member in Θ , implies that μ either (i) Believes p is currently false and wants it be eventually true (i.e., p is a *normal achievement goal*); or (ii) Having privately discovered p to be achieved, unachievable or irrelevant, μ has committed to having this private belief become Θ 's mutual belief.

Two important issues should be noted. First, there is a change in expressiveness of plans — in this framework, an entire team can be treated as jointly committing to a team plan. For example, when a company of helicopters flies to a waypoint, it is a team jointly committing to a team activity — each individual is not flying on its own to that waypoint, while merely coordinating with others. Thus, it is sufficient if the team reaches the waypoint, each individual need not do so individually⁴. Such a change in

³ $JPG(\Theta, p)$ also includes a common escape clause q , omitted here for the sake of brevity.

⁴This may mean that the first or some pre-specified percentage of vehicles reach close to the waypoint.

plan expressiveness alleviates concerns such as the fourth item in Figure 3.

Second, to establish a joint intention, agents must hold a WG (weak goal) which ensures that members cannot freely disengage from their joint commitment at will. In particular, while a $JPG(\Theta, p)$ is dissolved when a team member μ privately believes that p is either achieved, unachievable or irrelevant, μ is left with a commitment to have this belief become mutual belief. To establish mutual belief, an agent must communicate with other team members. While this communication is an overhead of team activity, it enables an individual to ensure that its teammates will not waste their time or face risks unnecessarily. This alleviates difficulties such as the second example in Figure 3, where an individual disengaged from the joint commitment without informing other team members, and exposed them to unnecessary risks.

This framework provides an underlying model of teamwork, enabling flexible reasoning about coordination activities. For instance, there is an explicit justification for communication, enabling agents to reason about it. The following now presents some key theoretical modifications to the framework to accommodate the complexities of real-world combat simulations.

3.1 Modifying Commitments

Fulfilling the requirements in $WG(\mu, p, \Theta)$ requires a team member to unconditionally commit to communicating with other team members, whenever it drops p as a normal achievement goal. However, in synthetic battlefields communication can be costly, risky or otherwise problematic. For instance, communication may break radio silence, severely jeopardizing a team's overall joint activities. Therefore, the unconditional commitment to communication is modified to be conditional on communication benefits to the team outweighing costs (to the team). Also included in this modification is an agent's commitment to search for alternative lower-cost methods of communication (e.g., the agent may travel to personally deliver the message, if using the radio is risky). Nonetheless, in some cases, benefits will be outweighed by costs, and hence no commitment to communication will result. In other extreme cases, an agent may be simply disabled from communication even after dropping its normal achievement goal (e.g., a pilot may be shot down).

Such communication difficulties require that other team members take up some of the responsibility for attaining mutual belief. In particular, a team member must attempt to track the team's beliefs in the status of their joint goal. For instance, if a company of helicopters reaches a well specified waypoint, the team can be tracked as recognizing its achievement, and thus unnecessary message broadcasts can be avoided.

A second modification focuses on the dissolution of a joint commitment (JPG). In particular, currently, if an individual μ is known to drop the normal achievement goal, the joint commitment is automatically dissolved. Yet, such an automatic dissolution is often inappropriate. For instance,

if one helicopter μ in the company of eight is shot down during an engagement, the helicopter company does not automatically dissolve its joint intention to execute its mission; that would waste the team's jointly invested efforts in the mission and render the company highly ineffective in combat. Therefore, if a team member μ is known to drop its normal achievement goal, the JPG's dissolution is modified to be conditional on: (i) μ 's role being critical to the continuation of the joint intention (as discussed in the next section); or (ii) pre-specified conventions. However, if μ communicates achievement, unachievability or irrelevance, then the JPG is dissolved as usual.

3.2 Complex Teams, Individual Roles and Failures

While not defined in terms of individual intentions, a joint intention leads individuals or subteams in the team to intend to do their "share" (role) of a team activity (subject to the joint intention remaining valid)(Cohen & Levesque 1991). In our work, a role constrains an individual or a subteam to undertake certain activities in service of the joint intention, and the role may vary with the joint intention.

One key issue here is that in complex teams, that involve multiple subteams, the success or failure of an individual's role performance does not directly determine the achievement or unachievability for the team's joint venture. As a result, an individual may succeed or fail in its role, yet communication may not necessarily result. Hence agents must communicate their role success or failures to other participants (should others be banking on this role performance). Furthermore, since agents may be unable to communicate (e.g., because costs exceed benefits), team members must track other agents' role performance. Based on information about others' role non-performance, team members can determine the viability of the team's joint intention or their own role. Two heuristics may be used:

1. *Critical expertise heuristic:* If the success of the team's joint intention is solely dependent on the role of an individual agent, then the agent's role non-performance (failure) implies that the team's joint intention is unachievable.
2. *Dependency heuristic:* If an agent's own role performance is dependent on the role of the non-performing agent, then the agent's own role performance is unachievable.

4 Implementing the Modified Joint Intentions Framework

To implement the modified joint intentions framework the concept of *team operators* has been defined. For the team Θ , a team operator OP will be denoted as $[OP]_{\Theta}$. The usual operators as seen in Figure 2 will henceforth be referred to as individual operators. As with individual operators, team operators also consist of: (i) precondition rules for selection; (ii) application rules (complex team operators

will lead to subgoals); and (iii) termination rules. However, unlike individual operators, team operators encode the expressiveness and commitments of joint intentions.

4.1 Team Operators: Expressiveness

Team operators express a team's joint activity rather than an agent's own activity. Thus, while individual operators apply to an agent's own state, a team operator applies to a "team state". The team state is an agent's (abstract) model of the team's mutual beliefs about the world, which include identities of members in the team, information about their joint tasks etc. For instance, for a helicopter company, the team state may include the routes to fly to the battle position. Figure 4 shows the new operator hierarchy of helicopter pilot agents where operators shown in boxes such as $[Engage]_{\Theta}$ are team operators (the non-boxed ones are individual operators). These team operators are not tied to any specific number of agents within a team.

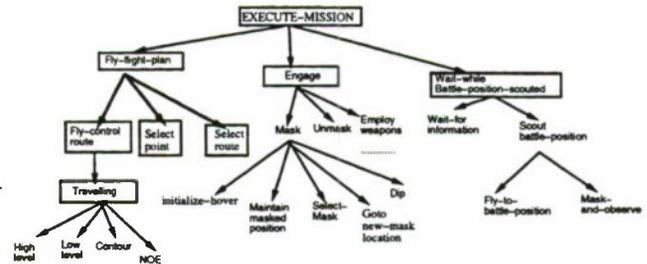


Figure 4: A portion of the new operator hierarchy, executed by an individual pilot agent.

To establish a joint intention $[OP]_{\Theta}$, each team member individually selects that team operator. Typically, this selection is automatically synchronized, since the selection is constrained by the team state (the team operator's preconditions must match the team state). Thus, since agents track their team state, visually and also via communication for terminating the previous team operator, it is usually unnecessary to explicitly communicate prior to the selection of the next team operator.

There are situations, however, where the agents' tracking of team states is not fully synchronized, so that their selection of team operators may be unsynchronized. In such cases, agents execute the "agree-and-execute" algorithm from (Kinny *et al.* 1992). In this algorithm, the leader (commander) broadcasts a message to all team members, seeking their commitment to the team operator to be executed next. Once it obtains everyone's commitments, it broadcasts another message to the team to begin the execution of that team operator. The execution of "agree-and-execute" is triggered if a team operator is executed in a subgoal after the completion of at least one individual operator in that subgoal. This ensures automatic synchronization among team members.

Note that in general, the subgoal of a team operator may lead to either a team operator or an individual operator to

be applied. Thus, a joint intention may lead to either another joint intention or to individual intentions in a subgoal (subject to the parent joint intention remaining valid). For instance, while the children of Engage_Θ are all individual operators, the children of $\text{Fly-flight-plan}_\Theta$ are all team operators.

4.2 Team operator: Communication

Once selected, a team operator can only be terminated by updating the team state (mutual beliefs) to satisfy the team operator's termination rules. Updating the team state may lead to a communicative goal. In particular, if an agent's private state contains a belief that makes a team operator achieved or unachievable, and such a belief is absent in its team state, then it automatically creates a communicative goal, i.e., a communication operator. When executed, this operator leads the agent to broadcast the information to the team. For instance, suppose the team is executing Engage_Θ , which is achieved if the team state contains the belief $\text{Completed}(\text{Engagement})$. Now, if a (commander) pilot agent's own state contains $\text{Completed}(\text{Engagement})$, and this is absent in its team state, then a communication operator is proposed to inform team members (the commander cannot just head back to home base alone).

To alleviate communication costs, certain safeguards are already built into the proposal of a communication operator. In particular, a communication operator is not generated if the private belief does not contribute to the achievement or unachievability of any active team operator, or if the team state is already updated, i.e., the team is already aware of the belief. Furthermore, based on the modifications discussed previously, even if a communication operator is proposed, it is not implemented immediately. Instead, the agent first evaluates the cost and benefits of the communicative operator. For instance, if radio is the current means of communication, and if the mission requires radio silence, communication over the radio is prohibited. An agent instead attempts to reduce communication costs via alternative communication methods, e.g., travelling to personally deliver the message. If the agent finally satisfies its communicative goal, the sender and the receivers then update their team state (we assume that communicated information reaches other agents securely). This then causes the team operator to be terminated (either because it is achieved or unachievable). If a high-level team operator is achieved or unachievable, its children are automatically assumed irrelevant.

4.3 Team Operators: Roles, Failures and Recovery

For team operators, roles are instantiated via suboperators in the operator hierarchy. If an OP_Θ has \mathcal{R} roles, denoted $\text{OP}_\Theta < \gamma_1, \dots, \gamma_R >$, then Θ 's R sub-teams, $\sigma_1 \dots \sigma_R$, must undertake each of these roles. Many team operators, however, can be defined via multiple role combinations. For instance, Engage_Θ may be performed by anywhere

between two to eight agents, some of them attack helicopters and some scouts. A separate representation of $\text{OP}_\Theta < \gamma_1, \dots, \gamma_R >$ for each role combination would result in a large number team operators.

To alleviate this concern, constraints are specified to only implicitly define role combinations. For instance, for Engage_Θ , the constraints specify that the allowable role-performing subteams are individual team members, i.e., the role performing subteam $\sigma_i = I$ where $I \in \Theta$; without any constraints on the number of participants. Each agent instantiates the constraint relevant to itself, to know if it is expected to act alone or as part of a subteam. The actual role an agent undertakes is based on this allowable subunit, and any static specification of the subunit's role in the current situation (e.g., an agent may be specified to be a scout). This role specification is in turn based on the subunit's or individual's capability. For a company of helicopters, a specific individual may be the commander (capability depends on the chain of command), a scout (capability depends on training), or the leader of a formation (every team member possess this capability).

As mentioned earlier, it is useful for an agent to monitor other agents' role performance. This is accomplished in one of three ways. First, the other agent may itself communicate. Second, it is possible to track the other agent's role performance, via techniques such as RESC (Tambe & Rosenbloom 1995; Tambe 1996; 1995; Tambe & Rosenbloom 1996), that dynamically infer other agents' higher-level goals and behaviors from observation of that agents actions. Given its expense, however, such detailed tracking is performed selectively — instead, an agent often only monitors the participation of other team members. Third, other heuristics can also be applied, e.g., an agent cannot perform two conflicting roles simultaneously. Thus, if a scout is scouting the battle position, it cannot participate in any other role at the holding area (e.g., to fly in formation).

The following describes the overall recovery algorithm, should an agent determine that $\mu \in \Theta$ is simply unable to perform any role (e.g., μ 's helicopter crashes):

1. Let $\mathcal{R} = \{r_1, \dots, r_N\}$ be the set of currently known roles of μ .
2. For each OP_Θ in currently active hierarchy and for each $r_i \in \mathcal{R}$ apply *critical expertise heuristic* to determine if OP_Θ unachievable.
3. If some OP_Θ unachievable, due to critical role r_c
 - (a) Terminate OP_Θ and its active children.
 - (b) If self capable of performing r_c , Communicate takeover of r_c to Θ ; Re-establish OP_Θ .
 - (c) If self incapable of performing r_c , Wait for another agent to takeover r_c ; Re-establish OP_Θ . If wait too long, OP_Θ unrepairable.
4. For each $r_i \in \mathcal{R}$ apply *dependency heuristic* to determine if unachievable; apply domain-specific recovery strategies.
5. For all $r_j \in \mathcal{R}$, $r_j \neq r_c$, If self capable of performing r_j , Communicate takeover of r_j to Θ .

- While μ disabled from performing any roles, check every future $\boxed{\text{OP}}_{\theta}$ via critical expertise heuristic.

One key reason this recovery procedure works is the explicit representation of team operators. In particular, step 2 applies the *critical expertise heuristic*. To operationalize this heuristic, the agent compares the achievement condition of an $\boxed{\text{OP}}_{\theta}$ with the achievement condition of μ 's role. If identical, μ was solely responsible for achievement of $\boxed{\text{OP}}_{\theta}$, and hence $\boxed{\text{OP}}_{\theta}$ is now unachievable. Thus, if μ is a scout, this test indicates that it is critical to the scouting of the battle position. In Step 3-a, the agent terminates $\boxed{\text{OP}}_{\theta}$ only if μ plays a critical role in $\boxed{\text{OP}}_{\theta}$. In step 3-b, the agent attempts to substitute itself for μ 's critical role if capability exists, or else it waits for someone else to fill in the role (step 3-c). Otherwise the implicated $\boxed{\text{OP}}_{\theta}$ is irreparable.

In step 4, the agent attempts to recover from any individual operator dependencies (step 4). Here, to operationalize the *dependency heuristic*, the agent checks the achievement condition of its own role for μ 's role. For instance, if an agent is to trail μ in formation, its achievement depends on μ . Non-critical roles are examined later, as they may be critical in the future (step 5). It is possible that one agent does not possess all of μ 's capabilities, and hence may takeover only one of μ 's roles, while other agents takeover μ 's other roles. Not all of μ 's roles may be known immediately; and hence any new operator is also checked for critical dependency on μ (step 6).

Interestingly, aspects (step 3-b and 3-c) of the above recovery procedure can also be cast as team operators, such that the communication in step 3-b and 3-c arises as a result of achieved or unachievability status of the team operator.

To see the above procedure in action, consider a company of five helicopters, Cheetah1 through Cheetah5, with the role and capabilities as shown:

Current roles:

Cheetah1 \leftarrow Commander, Scout
 Cheetah2, Cheetah3, Cheetah4, Cheetah5 \leftarrow Attack

Current capabilities:

Cheetah1, Cheetah3 \leftarrow Scout
 Cheetah2, Cheetah4, Cheetah5 \leftarrow Attack
 Chain of command: Cheetah1 \rightarrow Cheetah2 \rightarrow Cheetah3...

Suppose, the team is currently executing $\boxed{\text{wait-while-bp-scouted}}_{\theta}$. In service of this team operator, the scout (Cheetah1) is moving forward to scout the battle position, while the rest of the company is waiting at the holding area. Now if the scout crashes (as in Item 1 in Figure 3), $\boxed{\text{wait-while-bp-scouted}}_{\theta}$ is deemed unachievable (critical expertise heuristic). Two changes will then take place. First, Cheetah3 will take over the critical role of the scout — it has the capability of becoming a scout. This enables the $\boxed{\text{wait-while-bp-scouted}}_{\theta}$ operator to be re-established for execution. Next, Cheetah2, the next in command, will replace Cheetah1 as the commander.

5 Experimental Results

Agents based on our new approach each currently contain 1000 rules, with roughly 10% rules dedicated our explicit model of teamwork. This new implementation addresses three basic types of problems seen in our previous implementation:

- Recovery from incapacities of key individuals, such as a commander or a scout (e.g., addresses Item 1, Figure 3).
- Better communication and coordination within the team, as members recognize responsibilities (e.g., addresses Items 2 and 3, Figure 3).
- Improved tracking of own team state due to improved expressiveness (e.g., addresses Item 4, Figure 3); also possible to track team's high-level goals and behaviors, not possible before.

Figure 5 illustrates that our current implementation provides significant flexibility in the level of coordination among team members. The figure attempts to plot the amount of coordination among team members (y-axis) over simulation time (x-axis). The percentage of team operators in a pilot agent's operator hierarchy (which consists of team and individual operators) is a rough indicator of the amount of coordination. In particular, a lower percentage of team operators implies a higher percentage of individual operators and hence low coordination among members; while a higher percentage of team operators indicates tighter coordination. Time is measured in simulation cycles, with 9475 cycles in this run.

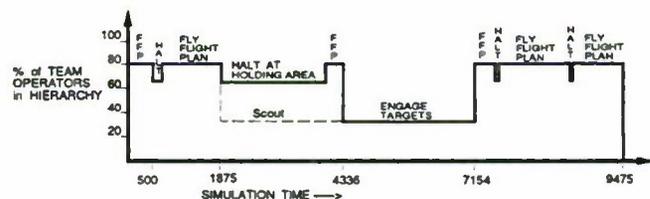


Figure 5: Percent team operators in an individual's operator hierarchy (FFP = Fly Flight Plan).

The varying percentage of team operators over the run indicates the flexibility in the level of coordination. Thus, for the first 500 cycles, when the agents are flying a flight plan (FFP) in close formation, they are tightly coordinated, an individual's operator hierarchy has 80% team operators. For the next 50 cycles, the company halts, and then resumes flying its flight plan. At cycle 1875, the company reaches the holding area, where the scout files forward to scout the battle position — the scout's percentage is shown separately by a dashed line. Basically, the scout is now only loosely coordinating with the rest of the company (33% team operators). After scouting, the company moves the battle position at cycle 4336, and until cycle 7154, engages targets. The 33% team operators in engaging targets indicate that the team members are to a large extent acting independently. Nonetheless, the team operator percentage

is never zero, i.e., these agents never act completely alone. Later the company returns to base.

Figure 6 illustrates the reduction in communication due to our modifications to the joint intentions framework. It shows results from a single test run of our implementation. Figure 6-a projects percentages of operators, had the agent worked with the original joint intentions framework. In this case, there are 25% team operators; and among the approx 75% individual operators, there are 25% communication operators and the rest execute the agents' actions. Figure 6-b shows the percentage from an actual run with the modified joint intentions framework. Communication percentage decreases more than 10-fold (just about 2% on communication). Instead, there is more emphasis on agent- and team-tracking, performed using RESC (Tambe & Rosenbloom 1995; Tambe 1996), with about 8% operators.

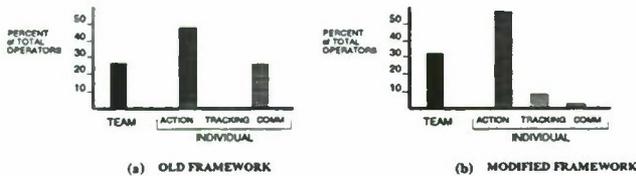


Figure 6: Reduction in percentage of communication operators.

6 Related Work

Few other research efforts have implemented theories of joint action. Jennings's implementation of the joint intentions framework in an industrial multi-agent setting is one notable exception (Jennings 1995). Huber and Durfee describe a similar implementation, although in a smaller scale testbed (Huber & Durfee 1995). There are several key differences in our work. First, in both these efforts, agents' collaborative activity appears to involve a two level hierarchy of a joint goal and a joint plan, with individuals engaged in specific roles in the plan. When the joint goal is accomplished, the collaborative activity is terminated. In contrast, our work focuses on complex, long-term team activities, involving the execution of a dynamically changing team operator hierarchy. A high-level mission leads to the execution of a whole variety team operators. It thus becomes essential to maintain and track an explicit team state, and manipulate it via team operators — else agents will lose track of the next team action. Second, the above efforts typically involve two-three agents in the joint intention. The scaleup from two-three agent to five-eight agent per teams (as in our work) creates new possibilities. More specifically, even if a single agent is incapacitated, the team operator hierarchy does not completely fall apart. However, agents have to explicitly check if lower-level team operators are unachievable, and recover from failures. Recovery is important, else the entire team effort will go to waste. Finally, in (Jennings 1995) issues of communication risk are not considered (although they are considered

in (Huber & Durfee 1995)).

Our recent work on team tracking (Tambe 1996) — which involves inferring other team's joint goals and intentions based on observations of their actions — is the predecessor to the work reported here. However, given its focus on tracking other teams, issues such as communication, recovery from unachievable team operators were all explicitly excluded from consideration. The domain of focus there was tracking the behaviors of a team of enemy fighter jets.

7 Summary and Discussion

For improved realism of advanced distributed combat simulations, teamwork among team members is critical. Yet, given the uncertainty in this domain, preplanned coordination cannot sustain such flexible teamwork. To alleviate this problem, we have provided individual agents with an explicit representation of team goals and plans, and an underlying explicit model of team activity, which has already substantially improved agents' flexibility in their teamwork. Further contributions of this paper include: (i) Detailed illustration of an *implementation* of the modified joint intentions framework (Cohen & Levesque 1991) in a real-world multi-agent domain; (ii) key modifications to the joint intentions framework to reflect important constraints in the domain; (iii) techniques for recovery from failure of team activities. As an important side-effect, agent development has speeded up, since once agents are equipped with such a model of teamwork, the knowledge engineer can specify higher-level *team plans*, and let the individual agents reason about the coordination activities and recovery.

The key lessons in this work are that as we build agent teams for increasingly complex combat simulations, agents should be provided (i) explicit representations of team activities, and more importantly (ii) some core common-sense knowledge of teamwork, separate from the agent's domain-level expertise (e.g., helicopter tactics). These lessons appears applicable to other agents within advanced distributed combat simulations, as well as to agents in non-military applications of advanced distributed simulations. Indeed, to test these lessons, we have begun implementing this framework for players in the RoboCup virtual soccer tournament (Kitano *et al.* 1995).

8 Acknowledgement

I thank Paul Rosenbloom, Jon Gratch and Randy Hill for discussions on teamwork. This research was supported under contract N66001-95-C-6013 from the Advanced Systems Technology Office (ASTO) of the Advanced Research Projects Agency (ARPA) and the Naval Command and Ocean Surveillance Center, RDT&E division (NRAD). Critical expertise and support has been provided by David Sullivan of BMH Inc.

References

- Cohen, P. R., and Levesque, H. J. 1991. Teamwork. *Nous* 35.

- Davis, R. 1982. Expert systems: where are we? and where do we go from here? *AI Magazine* 3(2).
- Grosz, B. J., and Sidner, C. L. 1990. Plans for discourse. Cambridge, MA: MIT Press. 417–445.
- Huber, M., and Durfee, E. 1995. On acting together: Without communication. In *Proceedings of the AAAI Spring Symposium on Reasoning about Mental states*.
- Jennings, N. 1994. Commitments and conventions: the foundation of coordination in multi-agent systems. *The Knowledge Engineering Review* 8.
- Jennings, N. 1995. Controlling cooperative problem solving in industrial multi-agent systems using joint intentions. *Artificial Intelligence* 75.
- Kinny, D.; Ljungberg, M.; Rao, A.; Sonenberg, E.; Tidhard, G.; and Werner, E. 1992. Planned team activity. In Castelfranchi, C., and Werner, E., eds., *Artificial Social Systems, Lecture notes in AI 830*. Springer Verlag, New York.
- Kitano, H.; Asada, M.; Kuniyoshi, Y.; Noda, I.; and Osawa, E. 1995. Robocup: The robot world cup initiative. In *Proceedings of IJCAI-95 Workshop on Entertainment and AI/Alife*.
- Levesque, H. J.; Cohen, P. R.; and Nunes, J. 1990. On acting together. In *Proceedings of the National Conference on Artificial Intelligence*. Menlo Park, Calif.: AAAI press.
- Newell, A. 1990. *Unified Theories of Cognition*. Cambridge, Mass.: Harvard Univ. Press.
- Rajput, S., and Karr, C. R. 1995. Cooperative behavior in modsaf. Technical Report IST-CR-95-35, Institute for simulation and training, University of Central Florida.
- Rosenbloom, P. S.; Laird, J. E.; Newell, A.; ; and McCarl, R. 1991. A preliminary analysis of the soar architecture as a basis for general intelligence. *Artificial Intelligence* 47(1-3):289–325.
- Tambe, M., and Rosenbloom, P. S. 1995. RESC: An approach for real-time, dynamic agent tracking. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*.
- Tambe, M., and Rosenbloom, P. S. 1996. Architectures for agents that track other agents in multi-agent worlds. In *Intelligent Agents, Volume II: Lecture Notes in Artificial Intelligence 1037*. Springer-Verlag, Heidelberg, Germany.
- Tambe, M.; Johnson, W. L.; Jones, R.; Koss, F.; Laird, J. E.; Rosenbloom, P. S.; and Schwamb, K. 1995. Intelligent agents for interactive simulation environments. *AI Magazine* 16(1).
- Tambe, M.; Schwamb, K.; and Rosenbloom, P. S. 1995. Building intelligent pilots for simulated rotary wing aircraft. In *Proceedings of the Fifth Conference on Computer Generated Forces and Behavioral Representation*.
- Tambe, M. 1995. Recursive agent and agent-group tracking in a real-time dynamic environment. In *Proceedings of the International Conference on Multi-agent systems (ICMAS)*.
- Tambe, M. 1996. Tracking dynamic team activity. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*.
- Tidhar, G.; Selvestrel, M.; and Heinze, C. 1995. Modeling teams and team tactics in whole air mission modelling. Technical Report Technical Note 60, The Australian Artificial Intelligence Institute.

Authors' Biosketches

Milind Tambe is a research computer scientist at the Information Sciences Institute, University of Southern California (USC), and a research assistant professor with the computer science department at USC. He completed his undergraduate education in computer science from the Birla Institute of Technology and Science, Pilani, India, in 1986. He received his Ph.D. in 1991 from the School of Computer Science at Carnegie Mellon University. His interests are in the areas of multi-agent systems, specifically agent modeling and multi-agent collaboration, as well as in parallelism and real-time performance of AI programs, especially rule-based systems.

Distributed Modeling of Cooperative Behavior by Mobile Agents

Peter S. Sapaty

Department of Electronic & Electrical Engineering,
University of Surrey, Guildford, Surrey GU2 5XH, UK
p.sapaty@surrey.ac.uk

1. Abstract

The paper is describing a project aimed at parallel and distributed modeling of cooperative behavior of intelligent autonomous entities. The simulation technology used is based on mobile cooperative agents which can freely migrate between computers and carry with them their states, operations to be executed and local data, thus providing the most natural mapping of dynamic scenarios with moving and interacting objects onto distributed computer networks. Some basic space navigation and mutual coordination mechanisms for cooperative behavior of distributed computer generated forces are described with their implementation in a compact agent-producing and coordinating WAVE language. A brief description of WAVE and a summary of application projects based on it are also provided.

2. Introduction

Computer generated forces (CGF) are both the heart and brain of a distributed simulation, and the level of their intelligence usually determines the intelligence and effectiveness of the whole simulation and training process. Of particular interest is the modeling of collective behavior of CGF, where group solutions of complex problems in a distributed space may be used for both friendly and opposing forces effectively complementing or competing with human-driven (simulated or live) entities. Due to the internal flexibility and self-organization, the latter often exhibit productivity which is much higher than a mere sum of their components, which explains the usual superiority of human teams over (distributed) computer models.

2.1 The Project's Objectives

The objective of the project described in this paper is to investigate and design new efficient methods of modeling and coordination of a collective behavior of intelligent autonomous entities propagating and

interacting in space and pursuing complex common goals. Research is being done into parallel and distributed algorithms showing various forms of collective behavior and having the following main characteristics:

- They should be implemented in a flexible distributed manner over a computer network and should be capable of exhibiting elements of robustness with respect to the failure of any number of computing nodes.
- The majority of interactions between entities should be at a local level: no centralized data base or controlling mechanism should be invoked for the basic operations within the groups of communicating entities.
- A multi-layer parallel, distributed and dynamic coordination structure should be designed, with automatic classification, pattern recognition and inferencing techniques on higher behavioral levels.
- Entities may incorporate elements of self-learning behavior and adaptiveness, leading if necessary to a considerable change of their own functionality, in order to improve performance with experience.
- Communication between entities should be minimized, remaining however sufficient to support main interaction patterns for collective operations.

2.2 Mobile Agents

A simulation technology chosen for this project is based in the ideology of mobile intelligent agents. Mobile agents have become a hot topic in recent years (Appleby & Steward 1994, Atkinson et. al. 1995, Gray 1995, Johansen et. al. 1994, Lingenau & Drobnik 1996, Di Marzo et. al. 1995, Ordille 1996, White 1994, and others) whereas code mobility is also an important factor in the latest development of conventional languages, like Java (Gosling & McGilton 1995).

As an extension of the client-server paradigm, an agent program can start from any computing node and travels independently through the network using the appropriate infrastructure. Such an infrastructure supports general services and openness of the network whereas local decisions, hops between computers, data processing and different kinds of interactions are carried out by the agent's program individually while visiting computing nodes along its itinerary. The currently pursued direction of these technologies is focused mainly on applications like remote information retrieval and electronic commerce (White 1994), where an agent is regarded as a monolithic autonomous program following some route. Agents can establish direct communication between themselves in some point (called a "meeting place"), or they can share information in the places while visiting them at different times.

2.3 WAVE

With a longer history than mobile agents and being functionally more diverse, is the WAVE paradigm (Sapaty 1988 and 1996, Sapaty & Borst 1996) which combines full mobility of special recursive code with dynamic creation and processing of arbitrary knowledge networks in a distributed space in a parallel pattern-matching mode. In WAVE, autonomous moving agents are produced only on the implementation level as a reflection of different branches of a recursive self-evolving navigational program. As integral parts of the same spatial algorithm, these agents may range from arbitrary large programs to elementary operations, and the WAVE language embeds special mechanisms for interactions between agents, between agents and the environment, and for the global control over their populations. Such control may also be fully mobile.

Experiments of using WAVE for simulation of dynamic systems (Sapaty et. al. 1994 and 1996) have shown its high flexibility for the creation of distributed virtual worlds and for organizing movement in these worlds of simulated entities with any interaction and coordination patterns between them. The latter may not be fixed in advance and can evolve in time and space. Having special spatial coordination mechanisms allowing control over societies of mobile agents, not present in other languages, WAVE may effectively complement other programming paradigms allowing migration of an interpreted program code, like Java (see Vuong 1996).

Based on full program code mobility, the WAVE paradigm can easily model existing mobile agent systems. As WAVE dynamically creates, activates and processes any graphs and networks arbitrarily distributed in computer networks, it can effectively implement any other computational models with the interpreted moving tokens and any functions in nodes, like petri nets, dataflow, actors, neural networks, object-oriented approaches, etc. (Sapaty 1996). Moreover, such WAVE-based networks may evolve in space and change their topology at runtime which other existing models usually cannot do. These features may enable WAVE to be used efficiently in distributed parallel inference, automated planning, machine learning, situation recognition and assessment, expert systems, network management, virtual reality, etc. All this may be quite useful in designing and implementing intelligent computer generated forces, with some ideas on that matter presented in this paper.

2.4 Further Structure of the Paper

The rest of this paper is organized as follows. In Section 3 elementary examples of programming in WAVE are shown and a brief description of the language is given in which main ideas of the paper will be subsequently demonstrated. Section 4 reveals basic mechanisms of navigation by mobile entity models of a discrete space represented, for simplicity, as a regular grid arbitrarily distributed between processors. In Section 5, examples of some higher-order space functions are described which may be used by moving and cooperating models for making autonomous decisions and organizing their group self-recovery after damages. Section 6 summarizes a number of projects written in WAVE which exhibit collective behavior of moving objects in distributed computer networks. Section 7 concludes the paper and is followed by acknowledgments, references and a short author's biography.

3. WAVE Language

WAVE treats any modeled world as a knowledge network (KN) with an arbitrary topology where nodes and links may associate with any information (as arbitrary long strings). This network may be arbitrarily distributed between processors and the topology of a computer network may correspond to or may be quite different from the knowledge network topology. Wave programs (also called *waves*) may process such abstract knowledge networks directly, regardless of their distribution in a physical computer network.

3.1 Elementary Programming Examples

3.1.1 Creation of Network Topologies

In Fig. 1, a WAVE program is shown which incrementally creates a simple network while physically moving in space and discarding its worked parts. In this program, CR stands for CReate rule, "#" is a hop operator with its left and right operands setting up a link-node pair to be passed (here created, in the context of the CR) during movement. "@" identifies an associative, or "tunnel", link to a node, "=" is an assignment operator, F is a moving, or frontal, variable accompanying the mobile wave. The latter carries an address of the created node "a" (symbolically shown in square brackets), to be accounted later in a cycle. A is an environmental variable always lifting address of a node in which wave currently stands, and period separates program parts which should be executed one after another. The CR rule, applied at the beginning to the whole program, is automatically inherited by the remaining mobile wave templates. The underlying interpretation system will distribute this network between as many processors as possible (here three), or to particular processors, if this is explicitly stated in the program.

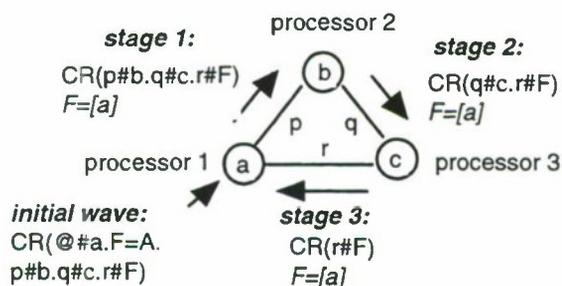


Figure 1: Distributed Network Creation in WAVE

3.1.2 Solving Tasks in a Navigation Mode

After creation, a KN (like the one above) may be navigated by other waves directly, to solve any distributed problem on it. For example, finding all direct neighbors of node "b" and printing their names in parallel on terminal(s) associated with these nodes, may be done as shown in Fig. 2a, where the rest of the program (or its "tail") replicates into two copies after broadcasting from "b" ("#" with missing operands means "any links" or "any nodes", or both, and results in local multicasting or broadcasting). T is a special read-write environmental variable which represents a terminal accessible from a KN node in which wave currently stands (the terminal may be different at different nodes), and C is another

environmental variable always lifting a content of the current node. While spreading in KN, wave transfers elementary operations to nodes which are executed in them using only local environment and variables associated with these nodes.

In Fig. 2b, a modification of this program is shown which first returns the neighbors' names back into "b" (using a predecessor address in the environmental variable P in both "a" and "c"), collects them as one list (in a shared, nodal variable N), and subsequently prints this list from "b". The SQ (SeQuence) control rule is used to activate the second branch of the wave only after full completion of the first one (which itself dynamically splits into parallel branches along links "p" and "q"). To simplify the picture, only data movement in frontal variables, and not the wave code it accompanies, is depicted. Character "," is used here as a separator between the two sequential branches.

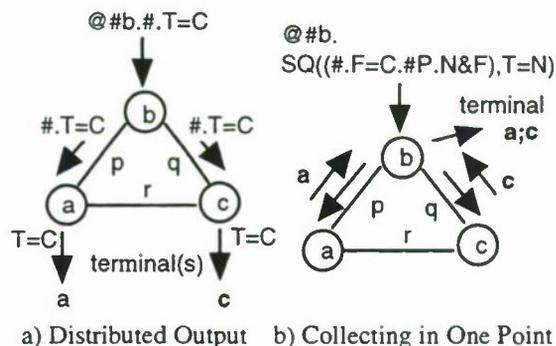


Figure 2: Printing All Neighbors of Node "b"

Different waves, originating from the same or different users, may be launched independently and in parallel on the same KN. For example, the two neighbors-printing programs shown above may be activated in parallel as:

(@#b.#.T=C), (@#b.SQ((#.F=C.#P.N#F),T=N))
 or
 @#b.(#.T=C), SQ((#.F=C.#P.N#F),T=N),

as they have the same starting node. Nodes, links or any variables may also hold procedures (as strings) which may be dynamically injected into main waves and executed. Waves may also modify the very KN they move through, while renaming, deleting and creating its nodes and links.

3.2 General Structure of the Language

The top level syntactic structure of a WAVE program, or wave, is shown in Fig. 3 where braces

mean zero or more repetitions (with a given delimiter at the right, if more than one), square brackets denote an optional construct, and vertical bar separates alternatives. Period delimits sequential parts and comma separates independent or parallel parts of a wave called *moves*. The latter may be simple, consisting of one or more elementary operations, or *acts* (like assignments, hops, condition checking filters, etc.) over information *units*, or again be waves in parentheses optionally prefixed by control *rules*. The latter impose a variety of constraints over distributed development of waves in KN. Starting from some node in KN, a move brings the wave into a new set of current nodes, or Goal Set, GS (which may include the initial one as well), to all of which the tail of wave is applied; waves thus being interpreted incrementally in KN. In general, many waves from the same program or from different ones may spread in KN in asynchronous wavefront mode.

```

wave -> { { move , } . }
move -> unit { act unit } | [ rule ] ( wave )

```

Figure 3: WAVE Language Recursive Structure

3.3 Basic Information Unit

The basic information unit of the language is *vector* – a dynamic sequence of arbitrary length values generally defined as strings, concrete interpretation of which depends on the operations involved. Syntactically, vector elements are separated by a semicolon. This simple data structure with special operations on it, together with the recursive syntax of the language, proved to be sufficient for representing arbitrary network creation and processing algorithms in a distributed environment. No explicit type descriptions are used in the language: automatic type conversions are activated depending on the operations.

3.4 Spatial Variables

Information units in a WAVE program can also be expressed by *spatial variables* dynamically distributed throughout KN by mobile waves. They may be of the three types: *nodal* (prefixed by N or M) dynamically attached to KN nodes and shared by different moving waves, *frontal* (prefixed by F) moving with waves and providing local information exchanges between different nodes of KN, and *environmental*, accessing currently available resources related to KN nodes and links. The latter are named as: C - node content, A – node address, L

– incoming link content, S – incoming link sign, P – predecessor node address, T – user terminal (or one of them if they are distributed throughout KN), and I – "individuality" (color) of the mobile waves (suffixing all their M-named variables).

3.5 Acts

Basic acts are selective or broadcasting *hops* in the KN (for which the two units keep information about links and nodes to be passed and also set up different sorts of local and global broadcasting), *condition checking filters* (halting if false), *data processing* (arithmetic and string operations), explicit *halts* with a repertoire of echoing termination conditions, and an *external call* permitting an access and exchange of information with other systems distributed in networks. Hops (the “#” act) identify by the left operand the links to be passed, and by the right operand the nodes these links should lead to (nodes may be given by contents or addresses). Omitting the right operand makes any destination nodes acceptable with the given links. Omitting the left operand leads to neglecting of the link contents and broadcasting to certain, if names are provided, otherwise to all, neighbors. The special name “@” used as the link operand triggers direct (tunnel) jumps between any (including non-neighboring) nodes, and provides broadcasting to all other nodes of KN if the right operand is empty. If more than one link with the given name is associated with a node, all of them may be passed in parallel.

Filters (“=” - equal, “/=”- not equal, “<” - less, “<=” - less or equal) allow for the further wave propagation if their result is TRUE, and cause halts if FALSE. Data processing includes arithmetic acts (“+”, “-”, “*”, “/”), splitting string into a vector (“!”) and merging vector into a string (“%”) with the given delimiters, appending vectors (“&”), finding/recording a content by an index (“:.”), finding an index by a content or recording by a content (“::”). Act “?” makes an access to other systems on the host (via its operating system) and “!” is a programmed halt with the operands establishing different halting conditions (right hand operand), or switching off the special control track mechanism while launching uncontrolled waves with an established life time (specified by the left operand). Act “=” means a mere assignment of the result obtained on the right to the variable on the left. In its absence, the result of the data processing operations is assigned to the leftmost unit (a variable) in the move. For example, N=N+F-1 is equivalent to N+F-1, thus making expressions more compact.

3.6 Rules

The main rules and their abbreviations are: SeQuence (SQ), Or Sequential (OS), Or Parallel (OP), And Sequential (AS), And Parallel (AP), RaNdom choice (RN), RePetition (RP), WaiTing (WT), InDivisible (ID), and CReate (CR). The rules split waves into branches (by their heads consisting of parts separated by comma or being single hops producing multicasting or broadcasting, with replicating and attaching common tail) and coordinate their cooperative (parallel or sequential) development in the KN (SQ, OS, OP, AS, AP), provide distributed logical synchronization (WT) and indivisible access to shared resources (ID), repeated application of the wave (RP), enable the wave to extend (including from nought) the KN it moves through (CR). The control points triggered by rules dynamically appear in different KN nodes and make distributed coordination of the propagating waves to a proper depth, using tracks with a variety of backwarded through them and merged control echo signals. The control points cease to exist after the termination of waves they oversee.

3.7 Dynamic Code Injection

It is possible to inject new strings into the moving wave as procedures (kept and processed as string contents of variables) which accompany the waves (in frontal variables) or are picked up in nodes of KN during navigation of the latter (by nodal and environmental variables). This provides high flexibility in the network creation and navigation processes where the evolving spatial program may be additionally fed from the distributed environment it moves through. Syntactically this is expressed by a move consisting of a single unit (a variable), without any act, which causes injection of its content into the wave with immediate execution of this code. Such injection may be recursive to an arbitrary depth.

3.8 WAVE Language Implementation

A distributed WAVE interpreter has been implemented in C and operates via Internet (Sapaty & Borst 1996). A copy of this interpreter must be installed in each computer, and the interpreters may communicate with each other while forming a parallel spatial machine driven by mobile waves. Parts of the KN and dynamic track forests (used for checking termination conditions of multiple distributed processes and also serving as logical channels for further spreading "waves" of waves), located in different interpreters, form together a

seamless distributed and dynamic information & processing space. In this space, waves (accompanied by moving data variables) and echoes (backwarded through tracks), are propagating either within the memory of the same machines, or are automatically forwarded to other interpreters on other machines.

4. Basic Distributed Coordination Mechanisms

In this section we will consider some basic space navigation and mutual coordination mechanisms for moving objects and their expression within the mobile agents philosophy. These mechanisms will be formally presented in WAVE language (with a preliminary explanation in English) as WAVE is a very dense machine-level language oriented on a direct interpretation in networked hardware within a program flow mode. As WAVE programs are moving in a physical space and parallel wave algorithms are highly communication-intensive, the program code must be very compact in order to reduce traffic in networks.

4.1 Representation of Space

As we are investigating here the use of mobile program code propagating between computers for simulation of dynamic systems with collective behavior, the main concern will be a discrete, rather than analog, model of space. The simplest one, however general enough to explain the main ideas of this paper, may be a regular grid. We will consider a two-dimensional grid with "x" and "y" coordinates, organized as a network of nodes having a combined "x-y" name each and connected with each other by oriented links named as "x" and "y" and directed towards the increasing values of these coordinates (see Fig. 4, where one of nodes 3-2, with incident links, is zoomed).

The WAVE model, oriented on a direct processing of arbitrary networks, can be easily used for creation and distribution of a regular grid between any number of processors, which may amount from one to the total number of grid nodes, while in the latter case each node may be located on a separate processor. The WAVE program for creating a 5 x 5 grid may look like the following (given here without explanations, only as an example of its compactness, as it is outside the main interest of this paper).

```
FX=5.FY=5.Fs=A.Fy=1.  
RP( SQ( ( FA=NA.Fx=1.N=Fx&Fy%-CR(@#N).  
      RP( FAN&A.  
        (CR(+y#FA:Fx.!3),.
```

```

      Fx==FX.@#Fs.NA=FAN.!3 ),
    ( Fx<FX.Fx+1.F=Fx&Fy%-
      CR(+x#F) ).!3 ),
    ( Fy<FY.Fy+1 ) )

```

4.2 Elementary Movement

Let us start with the most basic procedure: movement through the discrete space (regular grid, as we defined it). As mobile programs may be fully autonomous, self-contained objects carrying with them their states and local data (the latter hereinafter will be named with prefix "MOVING"), the movement through grid may be expressed as straightforwardly as:

```

Set up destination coordinates X and Y
Apply in starting node
Repeat
  Append current node to MOVING_PATH
  If current node is the destination
    print MOVING_PATH and halt
  otherwise
    If, by random choice between coordinates,
      it is possible to decrease distance to
      the destination, hop through
      the proper link to a new current node
    otherwise do nothing and stay in current node
endRepeat

```

During the movement (see Fig. 4), this mobile algorithm also accumulates and carries with it the passed path in MOVING_PATH which is output at the destination point. Looking at first sight like a conventional data processing procedure, this algorithm however operates only with the local environment connected to the current grid node it stands at, and changes its own location in space each time it moves to a new node, in which the whole Repeat-body is applied again.

The corresponding WAVE code will be as follows (where frontal variable FR is for MOVING_PATH, and the moving procedure separating "x" and "y" coordinates recorded in nodes with a hyphen between them is kept in frontal variable Fd):

```

FX=4.FY=2.@#'2-4'.
Fd='FD=C|- .Fx=FD:1.Fy=FD:2'.
RP( FR&C.Fd.
  OS( (Fx==FX.Fy==FY.T=FR.!),
    RN( ((Fx<FX.+x#),(Fx<Fx.-x#)),
      ((Fy<FY.+y#),(Fy<FY.-y#)) ), ) )

```

The randomly chosen and passed path will be printed in node 4-2, and may be as: 2-4;3-4;3-3;3-2;4-2.

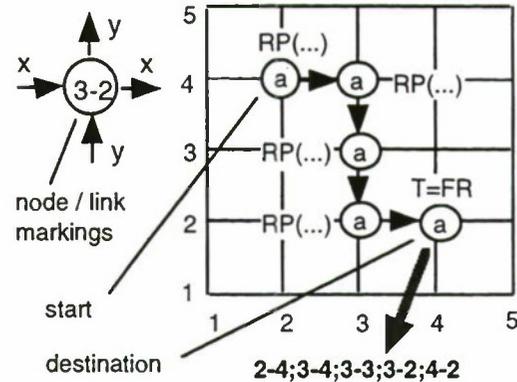


Figure 4: Movement through Space

4.3 Vision of Space to a Proper Depth

Another basic and very important procedure, while moving through space, may be vision of this space, say, to a proper depth, to assess the situation and make a proper decision. The following algorithm, starting from some grid node, dynamically creates a depth-first search tree to a given depth, in a maximum parallel mode, and then uses this tree for backwarding and stepwise assembling in one final list names of all the objects seen (together with names of the nodes they occupy). The algorithm is based on a self-invoking recursive MOVING_PROCEDURE, and the collected information is subsequently output at the start node.

```

Define MOVING_PROCEDURE as:
  Increment MOVING_COUNTER by 1 and
  continue only if MOVING_COUNTER is less
  than MOVING_LIMIT
  Hop to all neighbors in parallel
  If node is not marked, mark it
  otherwise halt this branch
  If there is an object in the node, put it together
  with the node's name into NODAL_LIST
  Do sequentially from the same node
    1) Apply MOVING_PROCEDURE
    2) Put NODAL_LIST into MOVING_LIST
  Hop to predecessor
  Append MOVING_LIST to NODAL_LIST
end MOVING_PROCEDURE
Begin in the proper start node
Put search_depth into MOVING_LIMIT
Mark current node
Do sequentially from the same node

```

- a) Apply MOVING_PROCEDURE
- b) Print NODAL_LIST

WAVE code for this algorithm, applied in node 3-4 and searching the grid for a depth of 2, will be as such:

```
F= 'Fd+1.Fd<=FD.#.N==.N=1.
  OS( ( Ni/=Nn=Ni&C%:' ), ).
  SQ( F, (F=Nn.#P.Nn&F) )'.
@'#3-4'.FD=2.N=1.SQ( F, T=Nn )
```

where MOVING_PROCEDURE is in F, and marking of nodes is performed by assigning 1 to the stationary, nodal, variable N in each node. It makes, as indivisible in each node, checking if the reached nodes are not marked yet and marking them (sequences of elementary condition checking and assignment operators are indivisible in WAVE and lock all nodal resources for other waves during their execution).

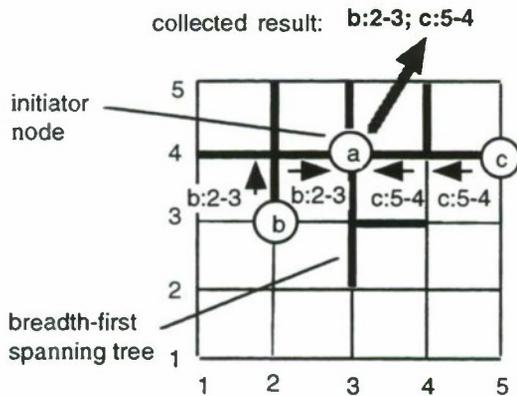


Figure 5: Breadth-First Parallel Space Search

In case of a success in a node, the program propagates further in parallel to all neighbors in which it is applied again. As operator "#" broadcasts generally to more than one node, the waves are self-replicating when navigating the network, passing nodes only once due to marks. This process is fully asynchronous and nondeterministic; however it always guarantees to receive a spanning tree covering the whole network. One of possible such trees is shown in Fig. 5 with the returned result to "a" as: b:2-3;c:5-4 or c:5-4;b:2-3, as the system is asynchronous and parallel.

4.4 Competition for Space

So far we were considering only one-object movement. Let us imagine that many objects are independently moving through the same space, and

let them not occupy the same grid node at the same time. So we come now to a concept of *competition for space*. When making a hop through the grid, now each object must win the right to occupy the new node, which may be described without details as:

- Find a prospective hop to a new node
- If the node is not marked,
 - mark it and move to this node
 - Hop to predecessor and remove the mark
- otherwise abandon this attempt

The WAVE program putting objects "a", "b", "c", and "d" into proper positions in the grid, each having the same destination 5-5, may be written as (having the main same body for each object, which will be automatically replicated):

```
WT( (@#1-5'.F=a), (@#2-5'.F=d),
    (@#2-4'.F=b), (@#3-5'.F=c).N=1 ).
FX=5.FY=5.
Fd='FD=C1-.Fx=FD:1.Fy=FD:2'.Fd.
RP( OS( RN( ((Fx<FX.+x#),(FX<FX.-x#)),
            ((Fy<FY.+y#),(FY<FY.-y#)),
            N==.N=1.(#P.N=!3), Fd ), ) )
```

Some possible snapshots of a collective movement of these four objects through a grid are shown in Fig. 6. As the destination node 5-5 may be occupied by only one object (here "b" succeeded first), others will be busy-waiting in the neighboring nodes forever, unless the objects are (self-) removed from the destination node, which may be programmed in their bodies.

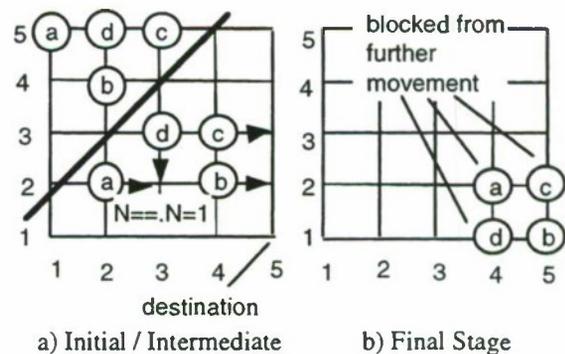


Figure 6: Competitive Movement

Having forbidden any two objects to be present in the same nodes simultaneously, we therefore established a minimum allowed distance between them in a grid as one. To make a two-step minimum distance the following should be done while moving:

Find a prospective hop to a new unmarked node
and mark it
If all neighbors of the new node
(excluding the current node) are not marked
Move to the new node
Hop to predecessor and remove mark in it
otherwise abandon this attempt and remove mark
from the new node

The corresponding cyclic part of the previous WAVE program should be rewritten for this case as:

```
RP( OS( RN( ((Fx<Fx.+x#),(Fx<Fx.-x#)),
            ((Fy<Fy.+y#),(Fy<Fy.-y#)).N==.N=1.
            OS( AS( AP(#.N==.!3),
                  (#P.N=.!3), Fd ),
                (N=.!4) ) ) ) )
```

Any other threshold distance between objects can be established, where the general solution may be based on the breadth-first space search to a proper depth described earlier.

4.5 Pursuit

Let us consider another basic scenario where one moving object (let us call it "pursuer") is chasing another moving one (or "escaper", not escapee), say, in order to destroy it. The escaper's mobile algorithm may generally look like:

Start from proper node
Put escaper's name into MOVING_NAME
Repeat
Mark current node by MOVING_NAME
Make time delay
If the node is not marked by MOVING_NAME
(i.e. marking changed by somebody else)
print "killed" in this node and halt
otherwise make next hop
endRepeat

If starting from node 2-2 with the escaper's name "a", and its route along the "x" coordinate towards the east, the wave may be as follows:

```
@ #'2-2'.Fi=a.
RP( Ni=Fi.2?sleep.
    OS( (Ni=Fi.T=killed.!3), (Ni=.+x#) ) )
```

Pursuer's full algorithm will combine repeated parallel space search to the allowed depth, followed with a hop towards reducing the distance between pursuer's current position and escaper's coordinates in the grid found in the search:

Start from proper node
Put pursuer's name into MOVING_NAME1
Put escaper's name into MOVING_NAME2
Repeat
Mark current node by MOVING_NAME1
Put current node address into
MOVING_ADDRESS
Do sequentially from the same node:
1) Repeat
Increment MOVING_COUNTER by 1
If MOVING_COUNTER is less than
search_depth and node is not marked,
mark the node
If node holds an object identified by
MOVING_NAME2
Return the node's coordinates to initiator
by MOVING_ADDRESS and halt
otherwise hop to all neighbors in parallel
endRepeat
2) Find, by a random choice between x and y
coordinates, a hop reducing distance to the
escaper and move to a new current node
endRepeat

With the pursuer's name "b", starting node 2-5, and escaper's name "a", WAVE code for this will be:

```
@ #'2-5'.Fi=b.Fesc=a.
Fd='FD=C1-.Fx=FD:1.Fy=FD:2'.Fd.
RP( Ni=Fi.Fs=A.
    SQ( WT( RP( Fc+1.Fc<4.N==.N=1.
              OS((Ni==Fesc.Fd.
                  @ #Fs.NX=Fx.NY=Fy.!3), #)).!3),
          OS( RN( ((Fx<NX.+x#),(NX<Fx.-x#)),
                  ((Fy<NY.+y#),(NY<Fy.-y#)).Fd), ) ) )
```

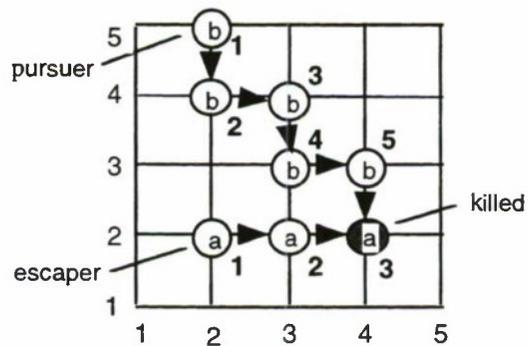


Figure 7: Pursuit of "a" by "b"

Taking into account that pursuer moves faster than escaper (the latter having an embedded time delay), it will eventually reach the escaper and kill it (actually, in the model, escaper discovers that his position is occupied by another object than itself, and self-

terminates). Different stages of movement of the both objects (numbered in bold> are depicted in Fig. 7.

4.6 Chaining by a Bilateral Agreement

Another than chase, common movement in space may be based on a mutual agreement between two objects where the first one, or leader, cannot move further unless the other, or follower, is close enough to it. The follower, in its turn, can move only if the leader releases its current position in space, to move into it. Leader's algorithm may be:

```

Define follower's name in MOVING_FOL
Make initial recording of the follower's address
Repeat
Do sequentially from the current node
1) Hop to predecessor node and wait until
   it becomes marked with MOVING_FOL,
   halt this branch
2) Perform time delay and make next grid hop
3) Lift control over this branch, make time delay,
   put current node's address into NEXT_HOP
   in the predecessor's node and halt this branch
endRepeat

```

This algorithm may be expressed by a wave with "a" as a leader, "b" as a follower, and the leader's and follower's starting nodes as 2-3 and 1-3 as:

```

Fi=a.Ff=b.Fs='2-3'.Fp='1-3'.
@#Fs.#Fp.#P.
RP( SQ( (#P.RP(Ni/=Ff).!3),
        (5?sleep.+x#),
        ( !1!3.1?sleep.#P.Nh=P) ) )

```

The follower may be expressed by the algorithm:

```

Define follower's name in MOVING_NAME
Start in proper node
Repeat
Mark current node by MOVING_NAME
Wait until NEXT_HOP becomes defined,
remove mark from the current node and
make grid hop by the recorded NEXT_HOP
endRepeat

```

With the follower's starting location in 1-3 and name "b", WAVE code will look like:

```

Fi=b.Fs='1-3'.
@#Fs. RP( Ni=Fi.RP(Nh==).Ni=.#Nh )

```

Two stages of the development of these cohesive moving processes are shown in Fig. 8.

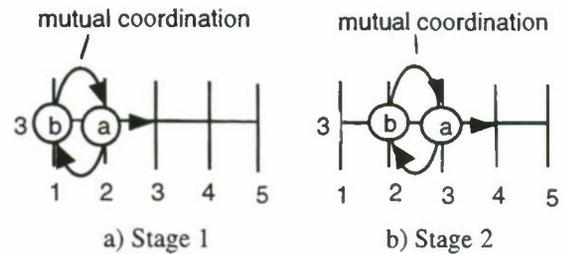


Figure 8: Bilateral Chaining while Moving

Any number of entities may be linked in a similar way to move cooperatively in space, with any spatial coordination patterns established and maintained between them. We will consider here, for simplicity, chaining objects as a column only, with the route being dynamically planned and followed by the column's leader. Within such a column, the first and the last objects will be organized the same as the leader and follower described above. But all intermediate objects will integrate some features of both leader and follower, as follows:

```

Define entity's name in MOVING_NAME and
the direct follower's name in MOVING_FOL
Make initial recording of the follower's address
Repeat
Mark current node by MOVING_NAME
Do sequentially from current node
1) Hop to predecessor node and wait until
   it becomes marked with MOVING_FOL,
   halt this branch
2) Wait until NEXT_HOP becomes defined,
   remove mark from the current node and
   make grid hop by the recorded NEXT_HOP
3) Remove NEXT_HOP record in current node,
   lift control over this branch, make time delay,
   put current node's address into NEXT_HOP
   in the predecessor's node and halt this branch
endRepeat

```

United WAVE code for "b" and "c" intermediate nodes, the main body of which will automatically replicate into two identical copies, with setting up of their names and starting positions, will be:

```

(Fi=b.Ff=c.Fs='3-3'.Fp='2-3'),
(Fi=c.Ff=d.Fs='2-3'.Fp='1-3').
@#Fs.#Fp.#P.
RP( Ni=Fi. SQ( ( #P.RP(Ni/=Ff).!3),
               ( RP(Nn==).Ni=.#Nn ),
               (Nn=.!1!3.1?sleep.#P.Nn=P) ) )

```

The column's movement is depicted in Fig. 9.

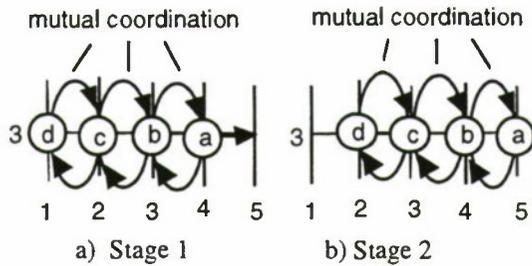


Figure 9: Multiple Chaining as a Column

Any other route, different from the above described which followed the "+x" direction only, may be set up in a leader. The others in the column will follow it exactly. Say, for following the route 4-4 -> 3-4 -> 2-4 -> 2-3, with the initial position in 4-3 and direct follower in 3-3, the leader should be organized as:

```
Frou='4-4';'3-4';'2-4';'2-3'.
Fi=a.Fs='4-3'.Fp='3-3'.
@#Fs.#Fp.#P.
RP( SQ( (#P.RP(Ni==).!3),
(5?sleep.Frou/=.#Frou:1.Frou:1=),
(1!3.1?sleep.#P.Nn=P) ) )
```

Two snapshots of the movement of the whole column along this route are shown in Fig. 10.

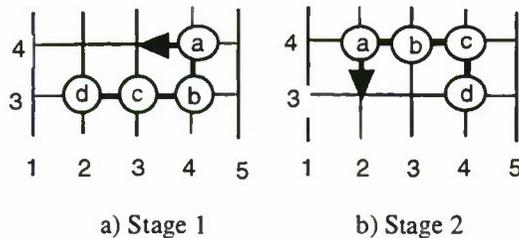


Figure 10: Arbitrary Route Followed by the Column

5. Some Higher-Order Functions

The described basic mechanisms of navigation and vision of a discrete space may be used for expressing higher-order distributed coordination functions of the cooperative behavior of computer generated forces. They may include, for example, election of a leader of a distributed team, spreading commands from leader to other team members, accumulating and averaging the knowledge acquired by different team members, etc. They may also include recognition of complex situations on distributed battlefields to make autonomous decisions and compete, say, with human-controlled simulated entities and their teams.

5.1 Pattern Recognition

We will consider here recognition of simple situations represented as structures distributed throughout the grid by using mobile programs. The latter may be launched from any moving entity (being itself a mobile program) while replicating into multiple copies and navigating in and searching the discrete space in parallel.

5.1.1 Group Recognizer

Assuming that objects distributed throughout a grid are considered linked with each other only if they are located in neighboring nodes, the following parallel algorithm collects into one list all objects linked by such neighborhood relation to an arbitrary depth, finding all such groups and printing the corresponding lists in parallel. As the same group may be recorded from any starting node, it will be allowed to be collected and printed as a list only if its starting node (kept in MOVING_START) is of the smallest (or biggest, as another variant) value in comparison with all other nodes, thus solving the competition in this simplest way and preventing the issuing of duplicates of the lists.

Define MOVING_PROCEDURE as:

```
If node is occupied
  Do sequentially from current node
  If occupant's name is less
    than MOVING_START
    halt with killing all processes
    originated from the start node
  If node is not marked, mark it by putting
    the occupant's name into NODAL_LIST
  otherwise halt
  Do sequentially
  1) Apply MOVING_PROCEDURE
  2) Put NODAL_LIST into MOVING_LIST,
    hop to predecessor and append
    MOVING_LIST to NODAL_LIST
end MOVING_PROCEDURE
Start in originator
Do sequentially
  3) Hop to all grid nodes, each becoming a start
  Continue if node is occupied, put occupant's
  name into MOVING_START
    and NODAL_LIST
  Do sequentially from a current (start) node
  a) Apply MOVING_PROCEDURE
  b) Append NODAL_LIST to predecessor
    node's FINAL_RESULT
  4) Output FINAL_RESULT
```

WAVE code starting from originator node 2-3 and then activating all grid nodes in parallel will be:

```
F= '#.Ni/=SQ( (Ni<Fs.!0), ).N==.N=Ni.
SQ( F, ( F=N.#P.N&F ) )'.
@'#2-3'.SQ( ( @#.WT( Ni/=Fs=Ni.N=Ni.
SQ( F, (Fr=N%-.#P.Nr&Fr ) ) ),
T=Nr)
```

Result printed in 2-3 may look like:

p-m-f-o-j-c; l-d-i; n-g-k-b-a-h-e ,
as shown in Fig. 11, where the order in which different groups are recorded may be arbitrary as the program is not deliberately synchronized and operates in parallel ("- " is used as a delimiter within the group lists).

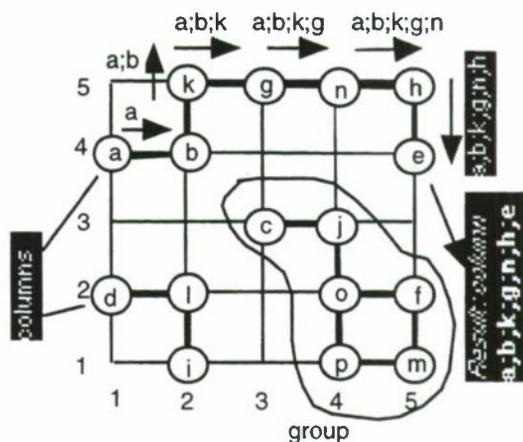


Figure 11: Recognizing Distributed Objects

Starting from all grid nodes in parallel, and also taking into account the necessity of individual nodal variables for the each growing chain solution (Mf and M are used), the corresponding WAVE code may be:

```
@#.Ni/=l=Ni.
RP( FR&Ni.
SQ( (#.Ni/=FR::Ni==.#P.Mf=P.M+1.!3),
( ( M==1.#Mf ),
( M==FR:-1<FR:1.T=Fr! ) ) ) )
```

The result will be:

a;b;k;g;n;h;e issued in "e", and d;l;i issued in "i" (see Fig. 11). With a slight modification of this wave, all results may be easily collected in the same node (as it was done in the previous example).

Any other structures and images, both deterministic and fuzzy, with any topologies and any distribution between machines, may be efficiently recognized by mobile wave agents recursively navigating the modeled space in parallel.

5.2 Self-Recovering Topologies

The WAVE model dynamically creating and processing arbitrary knowledge network topologies in a distributed environment may be efficiently used for organizing self-recovering control structures in which arbitrary failures may be repaired by the remaining parts of the control network. We will consider here a simplified case where an arbitrary graph node may be lost (together with all incident links), with the neighbors of this node discovering the damage and repairing it. After the repair, the recovered node is becoming a full member of the topology again, i.e. able to analyze the graph integrity and repair the damaged neighbors in its turn.

Start in all grid nodes

If node is occupied, continue

Repeat

Append occupant's name to MOVING_LIST

Do sequentially from current node

1) Hop to all neighbors

If node is occupied and

not in MOVING_LIST

Hop back to predecessor

Put the node's address into NEXT_HOP

Increment COUNTER and halt the branch

2) If COUNTER equals 1, hop by NEXT_HOP

If COUNTER equals 0 and first element in MOVING_LIST is greater than its last element, print MOVING_LIST and halt

endRepeat

The main idea is as follows. All nodes, regularly contacting their neighbors, copy the corresponding part of the topology, i.e. the star with themselves as the center, and select one of their neighbors as a "trustee", i.e. a node which will repair them (with the incident links) in case of damage, forwarding the copied star to the trustee. Each node periodically contacts the nodes to which it is a trustee, and in case of their absence creates the missing nodes with their links to other nodes, the latter represented by their addresses remembered in the recovery star.

It also loads a complete recovery program into the recovered nodes, making them active again and able to support their neighbors. All neighbors of the recovered nodes are informed about the recovery act and subsequently relaunch the copying of their stars and updating their trustees, as the address of the recovered node (assigned automatically by the underlying system) may have changed. Fig. 12 shows the situation when node "b" with its links to nodes "a", "c", and "d" has been damaged and later repaired by its trustee "a" which discovered the loss. (Addresses of nodes are symbolically shown as their names in square brackets.)

An active recovery template holding the copied star for a node, regularly renewed by every node itself (independently and in parallel with other nodes), is forwarded to its neighboring trustee and executed in the latter at regular intervals. It has the following structure:

- If the truster's node is missing
 - Create the node by its recorded name in the place automatically offered by the system or explicitly stated in the program
- Do sequentially from the created node
 - a) Create links leading from the node to other nodes represented by their addresses, mark these nodes to be self-activated later
 - b) transfer into the created node a full analysis & recovery procedure and activate it

WAVE code for this recovery template will be like:

```
OS( # node_name,
  CR( @# node_name.
    SQ( ( link_1 # node_address_1, ... ,
          link_m # node_address_m, N=1),
        ( !!. FP ) ) ) )
```

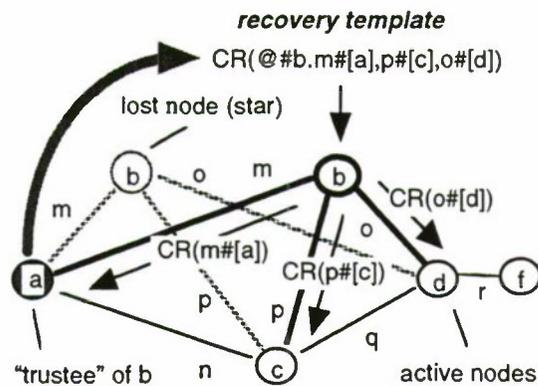


Figure 12: Automatic Recovery of "b" by a Trustee

General organization of the whole program making an arbitrary distributed graph self-repairing (any node at a time) without any central resources is as follows:

```
Define MOVING_PROCEDURE as
Repeat
Do sequentially
  1) If node is marked, remove the mark,
      create or update the recovery template
      for this node and record it in its trustee
  2) Activate recovery templates for all nodes
      for which the current node is a trustee
end MOVING_PROCEDURE
Start in all nodes, mark them, and
activate MOVING_PROCEDURE
```

The complete recovery WAVE program is as follows:

```
FP=
'RP( SQ( ( N/=N=.
  SQ( ( Ns=#.F=L&A%'#. #P.Ns&F %',
    ( F= 'OS( #' & C & ',CR( @#' & C &
      'SQ( (' & Ns & '.N=1),( !!.FP)))' ) ) %
    Fc=C.RN(#).
    OS( ( Nk::Fc==.Nk&Fc.Nt&F),
      ( Fk=Nk::Fc.Nt:Fk=F ) ) ).!3 ),
  ( Nt.!3 ), ) )'.
@#.N=1.FP
```

Being applied to all nodes of an arbitrary graph in parallel, it makes the graph undestroyable (with an assumption of any one node failure at a time). The program runs on any number of computers in a network, where the repaired nodes may migrate between them (eventually the whole self-recovering topology may migrate in such a way in a distributed environment after multiple node failures and recoveries). The described self-recovery spatial

mobile algorithm is extendible to the case where multiple nodes and links may be simultaneously damaged, while the whole network may self-recover in a completely distributed environment if at least a single node remains alive. This self-recovery technique, based on spreading of the recursive control code, may be useful for CGF and live command & control systems in many cases, for example, to maintain their organizational patterns in highly dynamic environments which can be damaged, say, in combat operations.

6. Mobile Programming Applications

We provide here a summary of some tasks effectively implemented in WAVE where mobile interpreted code migrating between computers has shown clear advantages before traditional stationary and compiled techniques.

6.1 Deaggregated Groups Clashes

Two groups of entities having different sources and destinations propagate independently through a large distributed grid and accidentally intersect in space, destroying each other's orders. The entities are not allowed to occupy the same node of the grid, whereas entities from one group (being "lightweight") keep a small distance from each other, and from another group (or "heavyweight") - a larger one. Lightweight entities are deaggregated from a single unit and interact with the heavyweight ones, aggregating later again having moved through the heavyweight orders. Each entity is represented by an independent WAVE program (into which the original program replicates) physically moving in a distributed environment.

6.2 Group Propagation Through a Maze

A group of entities generated at a source moves to a destination and meets on its way an arbitrary complex maze. Entities both compete for space and cooperate to break deadlocks in narrow corridors collectively. During their propagation the entities change between moving through free space, while reducing the distance to the destination, and going around walls. Deadlocks are resolved by self-changing the entities' functionality in a chain mode where, say, moving clockwise becomes dominant. Eventually all entities come to the proper destination.

6.3 Digging a Ditch Cooperatively

A group of entities (say, robots), starting from some point, moves towards a certain place and collectively

digs a ditch of a given size. Entities both compete with each other for space and work, and cooperate to ensure taking all the soil out, regardless of computer speeds and communication delays in a network. For this task, as well as for the others mentioned in this section, it is possible to create highly robust distributed models by using mobile programs which are free to move in space, can carry their states with them, and make complex decisions themselves.

6.4 Predator-Prey Games in Networks

A model has been written and shown in WAVE where a distributed computer network may be used for the analysis of behavior and pursuit of multiple moving objects in space. The objects may be injected into the network model at any time and from any nodes, and can have complex routes unknown in advance (say, being alien cruise missiles). They may be seen only from local nodes of a network, due to the limitations of physical (for example, radar) systems. Mobile waves in the implemented model were able to discover such objects and follow them in the network. The waves were also able to study behavior of the objects, measure and average their speeds throughout the network, subsequently organizing interactions between different types of objects with establishing dynamic predator-prey relations between them, with a variety of possible practical applications.

6.5 Dynamic Virtual Reality

Languages for representing 3D virtual reality in computer networks are of growing popularity nowadays, like VRML (Bell et. al. 1995). In an effort to add dynamics to VRML scenes, a number of its extensions have been developed. All these, however, base the description of a scene on a rigid hierarchical structure known as a "scene graph", which cannot be effectively changed from within the VRML programs as it reflects the structure of the program text. Successful experiments have been made using WAVE to provide fully distributed and highly parallel multi-user processing of VRML scenes, parts of which or the whole could be easily modified at runtime by treating scene graphs as WAVE knowledge networks, with parallel inference on them. The languages like VRML supporting visual representations of the modeled worlds and direct communications with the users were used on the surface of this semantic knowledge processing.

6.6 Distributed Dynamic Terrain

Standard visualization techniques have also been used on top of the dynamic distributed semantic worlds expressed in WAVE to model dynamic distributed terrain. For example, a possible representation of the terrain may be a regular grid again, each node containing data such as height, surface type, etc. Such a grid may be created in WAVE as a knowledge network (as described in Section 5) and dynamically distributed between any number of computers. Mobile wave societies have been created to produce on these grids actively changing shapes dynamically spreading among computers (e.g. growing craters, flooding, or "moving mountains"). This process is fully open, i.e. any (multiple) agent activities in these worlds can be started in parallel at any time, by different users, and from different machines. Other terrain representation, like, say, triangular networks may also be used.

7. Conclusions

On the results of the experimental programming presented and discussed in this paper we may say that mobile agent technologies may provide realistic models of collective behavior of autonomous entities which may efficiently operate in arbitrary computer networks, in a highly parallel mode, and without any central resources.

Mobile programming may be efficiently used on different levels of the description of a collective behavior of computer generated forces, ranging from elementary procedures of navigation and vision in a distributed space, to pattern recognition and assessment of dynamic situations distributed throughout computer networks. Mobile programming models may also exhibit high robustness and possibility of full self-recovery after complex failures in the underlying system software and hardware.

The obtained solutions for a discrete space as a regular grid may be easily modified and generalized for any other expression of a discrete or a combined discrete-analog universe. Representing fully distributed solutions, such mobile models may effectively compete with groups of human-driven (simulated or live) entities. They may also have a straightforward implementation in multiple live (including fully automatic) platforms, to be used on battlefields.

In this paper we have demonstrated a variety of mobile algorithms using the extremely compact WAVE language which self-migrates and navigates in

networks. Any other mobile agent techniques, as well as conventional languages like C, C++, Java, etc., may also be used for the expression of the presented collective behavior ideas, however programs in them may be up to 50-100 times longer and much more complex. The latter is because WAVE embeds special mechanisms of parallel navigation, interaction and coordination in space, with generalization of the distributed states (using tracks, for example) on a very high level. Being directly supported by a distributed WAVE interpreter, these features, however, have to be explicitly programmed for each application within most of the other techniques. As some support to this statement may also be the fact that the full machine programs for all discussed parallel and distributed space navigation and coordination mechanisms are included into this paper.

Further activities within the project described envisage the design of an intelligent high-performance wave chip from which any parallel and distributed self-organizing control structures may be networked, including the wireless teams of automatic platforms propagating and collectively solving complex problems in a distributed dynamic terrain.

8. Acknowledgments

This work has been partially funded by DRA Farnborough, UK, within a research project "Modeling Collective Behavior of Intelligent Autonomous Agents in Advanced Combat Operations". Special thanks are also due to James Darling, Malcolm Corbin, Peter Borst and Hans-Thomas Goetz for participation in this project and the discussions of the presented ideas.

9. References

- Appleby, S., and Steward, S. (1994). "Mobile software agents for control in telecommunication networks", *BT Technol. J.*, Vol 12, No 2.
- Atkinson, B., Brady, S., Gilbert, D., Levine, D., O'Connor, P., Osisek, D., Spagna, R., Wilson, L. (1995). "IBM intelligent agents", *Unicom Seminar on Agent Software*, London.
- Bell, G., Parisi, A., Pesce, M. (1995). "The virtual Reality Modeling Language. Version 1.0 Specification".
- Gosling, J., and McGilton, H. (1995). "The Java(tm) Language Environment: A White Paper". *Sun Microsystems, Inc.*
- Gray, R.S., (1995). "Agent Tcl: a transportable agent system", In T. Finn and J. Mayfield, Eds, *Proc.*

- CIKM'95 Workshop on Intelligent Information agents*, Baltimore, Maryland.
- Johansen, D., van Renesse, R., Scheidner, F. B. (1994). "Operating system support for mobile agents", In Y. Labrou, T. Finin, Eds, *Proc. CIKM'94 Workshop on Intelligent Information Agents*, Gaithersburg, Maryland.
- Lingnau, A., Drobnik, O. (1996). "Making mobile agents communicate", *Proc. etaCOM'96*, Portland, Oregon.
- Di Marzo, G., Muhugusa, M., Tschudin, C., Harms, J. (1995). "The messenger paradigm and its implications on distributed systems", in *Proc. ICC'95 Workshop on Intelligent Computer Communication*.
- Ordille, J. J. (1996). "When agents roam, who can you trust", *Proc. etaCOM'96*, Portland, Oregon.
- Sapaty, P. S. (1988). "WAVE-1: A new ideology of parallel processing on graphs and networks", *Future Generations Computer Systems*, vol. 4, North-Holland.
- Sapaty, P. S. (1996). "Mobile Processing in Open Systems", To be publ. in *Proc. HPDC-5 Symposium*, IEEE, Syracuse, New York.
- Sapaty, P. S., Corbin, M., Borst, P. M. (1994). "Using the WAVE Paradigm for Modeling and Control of Dynamic Multi-Agent Systems", *Artificial Life IV Conference*, MIT, Cambridge.
- Sapaty, P. S., and Borst, P. M. (1996). "WAVE: Mobile Intelligence in open networks", *Proc. etaCOM'96*, Portland, Oregon.
- Sapaty, P. S., Corbin, M. J., Seidensticker, S. (1996). "Mobile intelligence in distributed simulations", *Proc. 14th DIS Workshop*, Orlando, FL.
- Vuong, S., Ivanov, I. (1996). "Mobile intelligent agent systems: WAVE vs. JAVA", *Proc. etaCOM'96*, Portland, Oregon.
- White, J. E. (1994). "Telescript technology. The foundation for the electronic marketplace", White paper. *General Magic, Inc.*

10. Author's Biography

Dr. Peter S. Sapaty is Reader in Distributed Knowledge Processing at the Department of Electronic and Electrical Engineering of the University of Surrey, UK. His research interests include theoretical models and algorithms for parallel and distributed computing and control in open computer networks and their applications in system integration & management, artificial intelligence, distributed interactive simulation, telecommunications and dynamic virtual reality.

Authors List

- Adamson, Janusz M. - 49, 237, 493
Adkins, Michael L. - 313
Albright, Robert L. - 381
Alo', Richard A. - 573
Baker, Damon D. - 373
Balzer, Robert -181
Banks, Sheila B. - 101
Baxter, Jeremy W. - 319
Beheshti, Moshen - 573
Berkowitz, Jack - 131
Bimson, Kent - 57
Bombardier, Kevin C. - 291
Brooks, Wilbert J. - 127
Buettner, Cedric B. - 533, 555, 565
Calder, Robert B. - 19
Campbell, Chuck E. - 511
Carreiro, Richard L. - 19
Carver, Donald E. - 283
Chamberlain, Forrest - 533, 555, 565
Chandler, Edward V. - 355
Cisneros, Jaime - 255, 455
Clarkson, Jeff - 275
Cohen, Phil - 217
Coradeschi, Silvia - 93
Coulter, Karen J. - 203
Courtemanche, Anthony J. - 57
Craft, Michael A. - 141
Creech, Ross C. - 67
D'Urso, Robert - 209
de Korvin, Andre - 573
Dean, Christopher - 171
Durfee, Edmund H. - 329
Dymond, Marguerite M. - 127
Evans, Alan B. - 533, 555, 565
Fields, MaryAnne - 583
Fineberg, Michael L. - 479
Fischer, Pete - 435
Fisher, Jonathan - 533
Fogel, Lawrence J. - 265
Franceschini, Derrick J. - 159
Franceschini, Robert W.- 159, 427
Frosch, Ken - 441
Giguere, G.- 389
Goldman, Seth R. - 31
Gonzalez, Avelino - 171
Gratch, Jonathan - 37
Hartzog, Susan - 11
Hepplewhite, Richard T. - 319
Hieb, Michael R. - 243, 441
Howard, Martin D. - 419
Howard, Charles W. - 463
Howells, Peter B. - 389
Hu, Chenyi - 573
Hudson, Irwin L. - 347
Jackson, Greg - 79
Johnson, Thomas E. - 291
Jones, Randolph M. - 113
Joshi, K. G. - 237
Juliano, Michael - 209
Karlsson, Lars - 93
Karr, Clark R. - 141, 189, 255, 455
Kelly, Paul - 337
Kenny, Patrick G. - 329
Kluge, Karl C. - 329
Koc, Nazim - 119
Kocabas, Sakir - 119
Kraus, Matthew K. - 159
Kwak, Se-Hung - 397
Laird, John E. - 113, 203
Landry, Jean Philippe - 503
Laviano, Vincent P. - 441
Lu, Howard - 533, 565
Lyons, Reba - 397
Mabius, Larry - 533
Marshall, Henry - 355
Mastroianni, George R. - 367
McCauley-Bell, Pamela - 455
McClellan, Gene E. - 479
McEnany, Brian R. - 355
McKenzie, Frederic - 171, 511
Metzler, Ted - 151
Middleton, Victor E. - 367

Authors List

Miller, John - 79
Mullally, Daniel E. - 159
Mullis, Charles W. - 373
Naff, William T. - 283
Napravnik, Lee J. - 159
Nida, Jonathan C. - 427
Nielsen, Paul E. - 113
Nordyke, John - 151
Norwood, John D. - 303
O'Keefe, John A. - 463
Oviatt, Sharon - 217
Owen, Mark - 265
Oztemel, Ercan - 119
Panagos, James N. - 19, 291
Parsons, George M. - 283
Parsons, Rebecca J. - 255
Paz, Ben - 347
Peacock, Jeffrey C. - 291
Penney, Richard W. - 519
Peters, Steven D. - 479
Petty, Mikel D. - 67
Pittman, James A. - 217
Porto, Bill - 265
Powell, Edward - 209
Pratt, David R. - 3, 225
Pratt, Shirley - 225
Preston, Gary - 493
Pullen, J. Mark - 243, 441
Rajput, Sumeet - 189, 255, 455
Reece, Douglas A. - 337, 409, 471, 545
Salisbury, Marnie R. - 11
Santos, Eugene - 101
Sapaty, Peter S. - 599
Saucier, Paul - 463
Schricker, Stephen A. - 427
Shumaker, Gregory - 511
Siksik, Dave N. - 503
Sirisaengataksin, Ongard - 573
Smith, Ira - 217
Sousa, Mike - 533
Stanzione, Thomas - 533, 555, 565
Stober, David R. - 427
Stytz, Martin R. - 101
Tambe, Milind - 591
Tecuci, Gheorghe - 243
Thomas, Jr., John G. - 355
Tolley, Tracy R. - 159
Torne, Anders - 93
Tu, Hsiao-Kun - 545
Ullom, Lawrence - 435
Uludag, Mahmut - 119
Valade, S. - 503
Vrablik, Rob G. - 19
White, Elizabeth L. - 441
Wirthlin, Ralph - 409
Wise, Ben - 209
Yang, Tzu-Chieh - 217
Yi, John - 275