THESIS

ANALYSIS OF A 3-TIER DISTRIBUTED ARCHITECTURE FOR THE SECTOR ANTI AIR WARFARE CENTER

by

Matthew P. Howell

September 1997

Thesis Advisor: Luqi
Associate Advisors: Mantak Shing Michael Holden

Approved for public release; distribution is unlimited.
**Analysis of a 3-Tier Distributed Architecture for the Sector Anti Air Warfare Center**

**Title and Subtitle:**

The Marine Air Command and Control System (MACCS) is composed of a collection of legacy, stovepipe Automated Information Systems (AIS), each of which contain functionality which is widely duplicated throughout the MACCS. A proposed alternative architecture, the Common Air Command Control System (CAC2S), would leverage the investment currently being made in Command, Control, Communications, Computing, and Intelligence (C4I) systems which provide a robust set of functional services common to a wide range of mission critical applications. A plan for migration from the MACCS architecture to the CAC2S architecture is a required component for a successful transition.

This thesis describes the messaging and database methodology, the ongoing efforts to identify common data types and processes, and a proposed three-tier distributed object architecture, which will guide the MACCS migration to the CAC2S. A Software Engineering tool, the Naval Postgraduate School Computer Aided Prototyping System (CAPS), is used to model a component of the MACCS, the Sector Anti Air Warfare Center (SAAWC), in an effort to more precisely identify the critical data type representations and data processing requirements needed to properly specify the CAC2S.

As a result of this effort, a blueprint has been created to describe the methodology and analysis required to effect the migration from the MACCS architecture to the CAC2S vision.

**Subject Terms:**

CAPS, MACCS, SAAWC, CAC2S, Software Engineering, Rapid Prototyping

**Security Classification of Report:**

Unclassified

**Security Classification of This Page:**

Unclassified

**Security Classification of Abstract:**

Unclassified

**Report Date:**

September 1997

**Report Type and Dates Covered:**

Master's Thesis
ANALYSIS OF A 3-TIER DISTRIBUTED ARCHITECTURE FOR THE SECTOR
ANTI AIR WARFARE CENTER

Matthew Peter Howell
Captain, United States Marine Corps
B.A., University of Michigan, 1987

Submitted in partial fulfillment
of the requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
September 1997

Author: Matthew Peter Howell

Approved by: Luqi, Thesis Advisor

Mantak Shing, Associate Thesis Advisor

Michael Holden, Associate Thesis Advisor

Ted Lewis, Chairman
Department of Computer Science
ABSTRACT

The Marine Air Command and Control System (MACCS) is composed of a collection of legacy, stovepipe Automated Information Systems (AIS), each of which contain functionality which is widely duplicated throughout the MACCS. A proposed alternative architecture, the Common Air Command Control System (CAC2S), would leverage the investment currently being made in Command, Control, Communications, Computing, and Intelligence (C4I) systems which provide a robust set of functional services common to a wide range of mission critical applications. A plan for migration from the MACCS architecture to the CAC2S architecture is a required component for a successful transition.

This thesis describes the messaging and database methodology, the ongoing efforts to identify common data types and processes, and a proposed three-tier distributed object architecture, which will guide the MACCS migration to the CAC2S. A Software Engineering tool, the Naval Postgraduate School Computer Aided Prototyping System (CAPS), is used to model a component of the MACCS, the Sector Anti Air Warfare Center (SAAWC), in an effort to more precisely identify the critical data type representations and data processing requirements needed to properly specify the CAC2S.

As a result of this effort, a blueprint has been created to describe the methodology and analysis required to effect the migration from the MACCS architecture to the CAC2S vision.
# TABLE OF CONTENTS

I. INTRODUCTION ............................................................................................................. 1  
   A. BACKGROUND ........................................................................................................... 1  
   B. MOTIVATION  
      1. Messaging .......................................................................................................... 8  
      2. Database Technology ......................................................................................... 10  
   C. GOAL ....................................................................................................................... 11  
   D. SUMMARY OF CHAPTERS ..................................................................................... 12  

II. SURVEY OF C4I ARCHITECTURE ........................................................................... 13  
   A. CHAPTER OVERVIEW ............................................................................................ 13  
   B. DEFENSE INFORMATION INFRASTRUCTURE COMMON OPERATING ENVIRONMENT (DII COE) ......................................................................................... 13  
   C. GLOBAL COMMAND AND CONTROL SYSTEM (GCCS) .................................. 16  
   D. JOINT MARITIME COMMAND INFORMATION SYSTEM (JMCIS) '98 ............ 18  
   E. SUMMARY ............................................................................................................. 20  

III. SECTOR ANTI AIR WARFARE CENTER (SAAWC) FUNCTIONALITY .................. 21  
   A. CHAPTER OVERVIEW ............................................................................................ 21  
   B. SAAWC PROCESSES ............................................................................................ 22  
   C. SAAWC DATA TYPES ............................................................................................ 26  
   D. PRODUCERS .......................................................................................................... 29  
   E. CONSUMERS .......................................................................................................... 36  
   F. SUMMARY ............................................................................................................. 40
<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Figure 1</td>
<td>SAAWC Functional Processes</td>
<td>24</td>
</tr>
<tr>
<td>Figure 2</td>
<td>Broker Operator Drawn with xfig</td>
<td>43</td>
</tr>
<tr>
<td>Figure 3</td>
<td>Message Flow Table</td>
<td>45</td>
</tr>
<tr>
<td>Figure 4</td>
<td>CAPS SAAWC Prototype</td>
<td>47</td>
</tr>
<tr>
<td>Figure 5</td>
<td>Air Defense System Display Operator</td>
<td>56</td>
</tr>
<tr>
<td>Figure 6</td>
<td>Alert Server Operator</td>
<td>59</td>
</tr>
<tr>
<td>Figure 7</td>
<td>Broker Operator</td>
<td>61</td>
</tr>
<tr>
<td>Figure 8</td>
<td>Communications Server Operator</td>
<td>63</td>
</tr>
<tr>
<td>Figure 9</td>
<td>Correlation Server Operator</td>
<td>66</td>
</tr>
<tr>
<td>Figure 10</td>
<td>Data Management Server Operator</td>
<td>68</td>
</tr>
<tr>
<td>Figure 11</td>
<td>Device Server Operator</td>
<td>70</td>
</tr>
<tr>
<td>Figure 12</td>
<td>Map Server Operator</td>
<td>71</td>
</tr>
<tr>
<td>Figure 13</td>
<td>Message Server Operator</td>
<td>72</td>
</tr>
<tr>
<td>Figure 14</td>
<td>Name Server Operator</td>
<td>74</td>
</tr>
<tr>
<td>Figure 15</td>
<td>Print Server Operator</td>
<td>76</td>
</tr>
<tr>
<td>Figure 16</td>
<td>Security Server Operator</td>
<td>78</td>
</tr>
<tr>
<td>Figure 17</td>
<td>Time Server Operator</td>
<td>79</td>
</tr>
<tr>
<td>Figure 18</td>
<td>Track Server Operator</td>
<td>80</td>
</tr>
<tr>
<td>Figure 19</td>
<td>User-defined Data Type Hierarchy</td>
<td>83</td>
</tr>
</tbody>
</table>
ACKNOWLEDGMENT

The author would like to reserve his deepest gratitude for his wife, Susan, for her unwavering support while he pursued the goal of completing this thesis. He would also like to extend heartfelt thanks to his children, Clarissa, Tyler, and Wesley, for their devotion and enthusiasm, which provided a continuing source of inspiration during the course of this project. Finally, grateful appreciation is acknowledged to Professor Luqi, Professor Shing, and CDR Mike Holden for their analysis, guidance, and tireless contributions to the research and writing of this thesis.
I. INTRODUCTION

A. BACKGROUND

The Marine Air Command and Control System (MACCS) is composed of agencies, equipment, and the operating procedures required to provide the Aviation Combat Element (ACE) commander with the means to command, coordinate, and control all air operations within an assigned sector, and to coordinate those air operations with other Services operating in the battlespace. Recognizing the need to migrate from the current dissimilar, or "stovepipe", command and control (C2) systems prevalent in the MACCS to an open architecture which is compliant with the Department of Defense Information Infrastructure Common Operating Environment (DII COE), the Marine Corps has endorsed a Common Aviation Command and Control System (CAC2S) concept as a target system architecture. The CAC2S will be required to provide the automated aviation planning, situational awareness, decision aid, and execution tools currently available to MACCS operators, but to do so within an architecture which takes full advantage of the common messaging, database, network, security, and display services provided by a DII COE compliant Command, Control, Communications, Computers, Intelligence (C4I) Workstation.

B. MOTIVATION

The MACCS incorporates sensors, controls, weapon systems, and the agencies which employ these tools to accomplish their doctrinal missions. Common networks within the MACCS include a number of DOD standard datalink networks and air
command and control voice communications networks, all of which contribute to the development of a common Battlefield Awareness (BA) for a particular air/ground sector. Previous and current upgrades to the MACCS architecture have consisted of the incremental upgrading of MACCS components, primarily special purpose processors, display devices, and communications equipment. These components are required to meet a limited set of data and programming interface standards in order to ensure continuing interoperability. This development and fielding methodology has populated the MACCS inventory with a number of special purpose devices which are functionally sufficient for completing the MACCS mission, but are expensive to develop, procure, operate and maintain. Additionally, the common BA developed among MACCS operators working in the current architecture is constrained by the need to employ multiple "boxes" on the desktop, essentially "stovepipe" systems, each with a limited capacity for sharing critical battlespace information with other systems. Finally, MACCS operators must struggle in the current architecture to overcome the inefficiencies introduced by having to perform separate but related tasks in a number of different operating environments.

Traditional interpretations of the purpose of the MACCS focus on the goal of providing a Common Operational Picture (COP) of the Air Battlefield to the MACCS operator. This operator could be responsible for launching Homing All The Way Killer (HAWK) missiles, controlling aircraft as they transit certain air sectors, or preparing the Air Tasking Order (ATO) for the next day's sorties. The principal means for building this COP, as a means to developing superior BA, has been through the processing of
electronic contacts, both passive and active, into tracks, which are then assembled into a "track file." This track file is then distributed to terminals throughout the MACCS and to similar air operations and control centers in the other Services. Migration to a CAC2S architecture, which will use not only traditional sources of information to build BA, but also electronically generated mail, United States Message Text Format (USMTF) messages, and archived intelligence records, requires the development of a new concept. Such a concept could be envisioned as a virtual state machine, which builds BA through the rapid processing, correlation, storage, and retrieval of Air Battlefield information developed from both tactical and national resources. Each CAC2S terminal in turn will be a specific instance of this state machine, an instance which captures that portion of the Air Battlefield pertinent to that CAC2S operator's tasks.

Though many components of the MACCS, such as the HAWK missile system and the AN/TPS-59 radar system, will continue to be developed as special purpose computing systems, much of the functionality within the MACCS is specified for the development of superior BA to provide an environment in which the most efficient decisions with the greatest likelihood for success in the battlespace can be made. That is, a great majority of the tasks required of MACCS operators specify the receipt, processing, and analysis of information in order to implement decisions which effect the employment of plans, equipment, and personnel. In examining the migration of the MACCS to the CAC2S it is necessary to distinguish the components of the MACCS which sense and destroy enemy objects, such as the AN/TPS-59 radar and the HAWK missile systems, from those which
support the decision-making process. Once this distinction has been made, it will become possible to permit the design of an architecture which supports the vertical development of special purpose sensors and weapon systems in conjunction with the horizontal development of a general purpose information systems network which best supports the development of the MACCS operator's BA. The migration plan must recognize that the computing problems associated with sensors and weapon systems are different from the computing problems associated with decision-making tools and aids. The latter problems are well-researched, well-documented, and have solutions which are widely implemented in the commercial world. Moreover, within the DOD there is a tremendous amount of effort currently being devoted to the formulation of a definition of such a general purpose information systems architecture.

One of the principle benefits of military computing in the information age is to bring coherency to the COP. The introduction of DII COE Workstations and, in particular, the Maritime variant of the Global Command and Control System (GCCS), the Joint Maritime Command Information System (JMCIS), promises to bring to the MACCS common messages, data types, network communication protocols, display technology, security, administration, and database services. Implementation of these standards will significantly enhance the common BA of operators throughout the MACCS, as well as significantly decrease the costs associated with developing, procuring, fielding, maintaining, and training operators on new MACCS components. If perfect BA is the Holy Grail of the warfighter, and computing is certainly integral to that goal, then
common, sophisticated messaging and database technology must be the cornerstone of efforts to migrate legacy computing systems. The MACCS is an ideal candidate for the mapping of that migration path.

There are two clearly identifiable revolutions in military computing: the first revolution involved the realization that computing could be used to create abstractions of real-world concepts, and to manipulate those abstractions in ways which provided meaning and intelligence to the operator trying to understand the state of the battlefield. This revolution was realized on special-purpose, embedded systems hardware, and represented a coherent "system-engineering" approach to the specification, design, and implementation of these machines and networks. Examples of these manifestations of special-purpose systems surround us today in the military: the Tactical Digital Link (TADIL) processors with their associated display subsystems and communication equipment; the Automated Digital Network (AUTODIN) switching centers with their associated processing, display, and communication subsystems; the Tactical Receive Equipment (TREs) with their associated processing, communication, and cryptographic subsystems.

The second revolution in military computing, and the one in which we currently find ourselves, came about with the invention and rapid adoption of some key computing technology and protocols: the tremendously powerful and inexpensive microprocessor with its associated peripheral devices; and Transport Control Protocol/Internet Protocol (TCP/IP), a widely adopted set of internetworking protocols which transcends proprietary
network protocols, allowing heterogeneous computers to communicate and truly distribute processing in powerful ways. The first factor in particular forced computer scientists to focus less on "system engineering" and more on "software engineering." They discovered that computing machines and computing networks were rapidly entering the realm of commodities. General purpose computers and networks became so powerful and inexpensive that software engineers were forced to modularize their computing abstractions to take advantage of computing architectures, rather than computing systems.

Three trends have accelerated the movement to implement the computing systems of the first revolution within the computing architecture of the second revolution: decreasing amounts of money, increasing redundancy in written program source code, and decreasing complexity in administering systems. General purpose computers and networks powerful enough to manage and manipulate the computing abstractions of our mission critical computing systems are widely available in the commercial sector, and inexpensive enough to motivate the move away from many special-purpose embedded systems. Moreover, the software programs and processes required to meet the needs of many of our mission critical functions, such as messaging, financial management, and logistical operations, are increasingly available as well in the commercial sector.

The redundancy issue involves the tremendous amount of time and money invested in creating and maintaining the same computing algorithms, data sets, and objects over and over again throughout organizations and industries. Recognizing the expense of a typical line of code, which drives the need to craft reusable code modules with
well-designed interfaces, has forced software developers inside and outside of the DOD to focus on identifying the functionality and required data sets of the target domain, in order to enable the search for, and acquisition of, existing computing abstractions at a significant savings over traditional in-house custom-built computing solutions.

Finally, the trend toward decreasing complexity is one in which the complexity of the computer system, manifested in such computing domain concepts as network administration, runtime environment configuration, and peripheral device installation and management, without knowledge of which computing systems cannot be properly administered and maintained, is evolving toward increasing abstraction of these computing domain principles and mechanisms. This trend frees the user to focus on the missions and tasks for which he or she is trained, rather than on mastering the techniques and methods of operation created by software developers to facilitate the tasks and operations associated with the development of software systems.

The proceeding discussion is meant to illustrate the following point: we no longer have either the resources or the incentive to develop special-purpose, embedded computing systems for all but the most unique and time-critical of our computational needs. And in order to identify the commodity computing machines, networks, and software processes which best fulfill our computational needs, it is critical that we identify that mission essential functionality, and those required data sets, which represent and describe our targeted operational domains. This paper examines the processes and data types which embody the required functionality of the Sector Anti Air Warfare Center
(SAAWC), itself a subsystem containing much of the functionality exhibited in the MACCS. The accurate and thorough specification of the processes and data types within the SAAWC is the critical first step in developing an object-oriented, network-centric, standards-based, relatively inexpensive, BA system which meets the required functionality of the USMC, complies with the standards-based environment of the DII COE, and takes advantage of the key technologies of the second revolution in military computing: virtual machines, commodity computers and TCP/IP-based networks, distributed object computing, and object Relational Database Management Systems (RDBMS).

1. Messaging

Computers are particularly useful for two things: messaging and database operations. Messaging is as old as man himself and often serves as the principle reason behind success or failure on the corporate and military battlefields. Messaging drives decisions, which in turn drive actions. Messaging is the principal means for developing BA. Timely and accurate messaging separates winners and losers, victors and the vanquished. Messages need not themselves be sophisticated: a simple message identifying enemy actions at a particular point and place in time can turn a battle. Message transmission technology need not be sophisticated: a timely phone call, simple e-mail, or faxed image can convey immediate and pertinent information. What requires a concerted and orchestrated effort, however, and a sophisticated analysis and design as a preliminary stage, is the development of a messaging system which defines each message as one of a finite set of Abstract Data Types (ADT): readable, representable, and redistributable to
both humans and computers, and a means for transferring these ADT "objects" rapidly across distinct but interoperable networks.

It is important that a MACCS to CAC2S migration plan recognizes the significance of these ADT "objects", which are both the means by which BA is developed and the means by which tactical, operational, and strategic decisions are implemented. An aircraft "track", an Air Tasking Order (ATO), and an image of an air defense site do in fact have more in common with one another than not. It is critical to the success of the migration plan that the precise computational representations of each of these ADTs be standardized in order to determine the most appropriate means for their transmission, reception and routing.

Furthermore, strong consideration should be given to our representation of abstract events as objects which can exist in a sequence of varying states. For example, an ADT object which was instantiated as a collection of attributes representing a planned air sortie could go through a series of modifications in which the object's own methods modified its state, as a way of indicating the object's transformation from a planned mission, to an executing mission occupying battlespace and battletime, to a finished mission with associated attributes which indicate the consequences of the executed mission. By unifying the conceptual, documented, executing, and historical attributes of a particular event in time, and by encapsulating within the object the methods which can direct the transformation of the object in response to real-world activities, we produce ADTs with properties which facilitate their creation, dissemination, and use within a
distributed messaging system.

To illustrate with another example why deliberate analysis of the messaging process, as it relates to ADTs, is needed, a solution in the CAC2S must be developed to overcome the current limitations in wireless bandwidth which have been exposed in current implementations of the TADIL J processing system. That system has been developed to accomplish the desired refresh rate of the air defense picture through the periodic transmission of the entire battlespace by means of a "track file", providing static position and descriptive information which is associated with each individual track. A more efficient solution uses ADT methods to send messages updating ADT representations when changes to those representations are triggered. Furthermore, by treating the air defense picture as a composition of ADTs, current broadcast, multicast, and subscription messaging methodologies can be implemented to deliver just those ADTs of interest to a particular given user, reducing the amount of traffic over the TADIL J link.

2. Database Technology

Automated Information Systems provide the capability to store extraordinary amounts of data, to index in an extensive and thorough manner, and to retrieve data in powerful ways which will aid BA, problem analysis, and the decision-making process. Traditional hierarchical database methodologies required the database designer to build a database schema with a particular "view", or collection of attributes which composed an entity, in mind. The resulting database schemas purposefully planned for the duplication of database entries in order to account for the many different views different users might
require to the database. Relational database methodologies allow the database designer to build carefully organized relations between distinct, but related entities, allowing for the creation of multiple views into the database schema without the accompanying duplication of database entries.

The MACCS to CAC2S migration plan must include a comprehensive analysis of the data needs of MACCS operators in order to determine the types of information required, the relationships between those information types, and the points and places of replication necessary to permit and enforce the provision of a common BA throughout the MACCS. Efficient design and implementation of relational database technology will support the storage, archiving, and retrieval of perishable, time-critical messaging as well as the provision of encyclopedic data elements or developed information which supports the decision-making process. Relational database technology is the key to efficiently processing and presenting to the user the correct and pertinent COP, amidst the multitude of messages, information, and intelligence which may be available. The migration plan should seek to identify every decision-making point within the MACCS and to develop a plan for incorporating relational database technology into that decision-making process.

C. GOAL

There is significant value added in the analysis of data processing requirements and data flow requirements in the MACCS, by modeling the network nodes and links in the SAAWC using the Naval Postgraduate School (NPS) Computer Aided Prototyping System (CAPS). CAPS is used to decompose each of the SAAWC functional
requirements into functional operating nodes, then further decomposing them into common services nodes, and finally, identifying data types and atomic data processing requirements within the resulting nodes and links. The result is an executable model which will facilitate the identification of particular information producing, messaging, and consuming needs and which will suggest a blueprint for the fielding of general purpose C4I Workstations and general purpose networks to meet those needs.

D. SUMMARY OF CHAPTERS

Chapter II examines the current landscape within the USMC, USN, and DOD with regard to development of C4I Workstations and general purpose networks. Chapter III documents an analysis of SAAWC data types and data processing requirements and provides an illustration of how those requirements might be satisfied by an infrastructure of common C4I services. Chapter IV describes the design of the SAAWC prototype using CAPS to decompose the system into a network of data streams and operators. Chapter V provides conclusions, recommendations, and identifies areas in which further research is merited.
II. SURVEY OF C4I ARCHITECTURE

A. CHAPTER OVERVIEW

This chapter discusses the various initiatives and design decisions published by the three C4I program offices influencing the development of the CAC2S. These design decisions represent guidelines and criteria for C4I segment developers, as well as standards for the provision of testing and assigning ratings which describe the levels of compliance to the prescribed architecture that developed segments meet. The DII COE provides a high-level Information Systems (IS) architectural plan for the building of C4I, Sensor, Weapons, and Combat Support systems. These systems will provide an operational environment which optimizes the flow of data vertically through the levels of operation, and horizontally across peer services and agencies. Of the three programs, only the DII COE is not an actual system. The Global Command and Control System (GCCS) is a C4I system designed to incorporate core operational, intelligence, and communication planning and execution functionality, and is intended to be the target architectural environment for each of the service-specific C4I system variants. The Joint Maritime Command Information System (JMCIS) is the USN/USMC variant of the GCCS system, incorporating functionality unique to the maritime character of USN/USMC warfare.

B. DEFENSE INFORMATION INFRASTRUCTURE COMMON OPERATING ENVIRONMENT (DII COE)

[JOINT95] proposes a concept, C4I for the Warrior (C4IFTW), which calls for the development of a general purpose architecture in which users at the tactical, operational,
and strategic levels work within a common operating environment accessing shared data pertinent to the prosecution of their particular missions. The C4IFTW vision is articulated as follows: "The warrior needs a fused, real-time, true picture of the battlespace and the ability to order, respond and coordinate vertically and horizontally to the degree necessary to prosecute the mission in that battlespace."

The DII Master Plan is a blueprint for implementing the technical infrastructure, shared services, and functional applications facilitating interoperability and collaboration among the DOD Services, Agencies, Office of the Secretary of Defense (OSD), and Joint Staff in order to accomplish the C4IFTW concept. The DII itself can be described as a seamless, worldwide, secure, standards-based web of computing hardware, software, and communication links designed to meet the information processing needs of DOD users in peace and in time of conflict. The primary purpose of the DII Master Plan is to identify and document current and future elements of the DII which enable interoperability and collaboration, define the roles and responsibilities of those falling under the cognizance of the DII, and to identify and document the relationships among current DII initiatives.

That portion of the DII Master Plan responsible for defining the set of integrated support services and software development environment for the DII shared technologies is the DII Common Operating Environment (COE). The DII COE contains the detailed technical specifications which support the DII architecture in accordance with the Technical Architecture Framework for Information Management (TAFIM) and DOD Joint Technical Architecture (JTA). The DII COE is an evolving computer systems
architecture, a set of standards designed to take advantage of commercial sector
technology and methodology, and a vision to guide the development of C4I and non-C4I
mission domain computer systems which realize the C4IFTW vision.

The DII COE defines three layers: Kernel, Infrastructure Services (Data
Exchange), and Common Support Applications. The Kernel includes the computer
operating system and extensions, the common desktop, software install and de-install
tools, security extensions, and printer services. The Infrastructure Services layer, a
horizontal layer, identifies RDBMS servers/clients, Web servers/clients, network
management processes, message profiling, office automation, and PC services. Common
Support Applications, also called vertical market services, include Mapping, Charting,
Geodesy, and Imagery (MCG&I), communications input and output, message encoding
and decoding, correlation and fusion, and tactical data replication. Mission Applications,
or segments in DII COE terminology, are developed to run on top of these three layers in
accordance with the DII COE Integration & Run Time Specification, which defines the
manner in which an application is to interact with underlying layers and with other, peer,
mission applications. Additionally, this document identifies eight levels of DII COE
compliance which provide a benchmark for judging the qualification of a mission
application in meeting DII COE standards. The D6 directorate, Joint Interoperability and
Engineering Organization (JIEO), within the Defense Information Systems Agency
(DISIA), is the cognizant authority for developing and publishing the DII COE and
anticipates major releases of the standard being published every 18 months, with minor

C. GLOBAL COMMAND AND CONTROL SYSTEM (GCCS)

To use a computer programming metaphor, if the DII COE is the specification for a "class", then the GCCS is the instantiation of that class, or the object itself. The GCCS is an actual system, developed to fulfill the C4I/FTW vision as it pertains to the functions of Command, Control, Communications, and Intelligence. The GCCS was originally conceived as a replacement for the World Wide Military Command and Control System (WWMCCS), the mainframe-based system which has served the command and control needs of high-level US military commands for over two decades. It is now also targeted as the system which will satisfy the vision of the C4I/FTW concept.

In February, 1995, a GCCS Design Working Group was convened [BUTLER96] to specify, define, and publish an architectural style and a set of specific architectural components designed to satisfy both documented GCCS COE requirements and the GCCS baseline environment. A number of command and control scenarios were developed and analyzed in order to determine the behavior and interrelationships of the architectural components the group had identified. The group came to the conclusion, documented in [BUTLER96] and [MOXLEY96-1], that a traditional two-tier architecture was unlikely to provide the robustness and flexibility required in a computing environment envisioned to satisfy both real-time needs and non-real-time needs, processing tactical and non-tactical data, and incorporating both high-speed LAN technology and lower-speed
WAN technology in a single global C4I network. They proposed as an alternative a three-tier architecture incorporating a "subscription broker" middle tier to serve as a mediator between consumer-oriented processes (traditional clients) and producer-oriented processes (traditional servers). This design decision is significant because it has repercussions not only for GCCS, but for every GCCS variant, such as the USN/USMC JMCIS, and every GCCS variant segment, such as a JMCIS application to monitor and control the prosecution of air defense operations. The CAC2S must be specified and designed with this in mind.

A three-tier architecture supports several common software engineering principles. It supports modularity and encapsulation, or information hiding, by allowing for the development of clients and servers with no need for knowledge of the implementation details another module might use. This significantly enhances the independence with which clients and services can be developed. It minimizes the amount of required system-wide knowledge, which now consists of only data types and services, as described in [BUTLER96]. It also supports the coexistence of persistent and non-persistent data, which is critical to the concept of combining real-time and non-real-time data requirements, by providing a brokered structure which prioritizes the delivery of information.

In a GCCS three-tier-architecture, client applications are mission-specific data consumers which might support the decision-making processes of analysts and operators. The subscription broker might be implemented using the Distributed Object Management
(DOM) architecture and one of two, or both, DOM technologies: the Distributed Computing Environment (DCE) and the Common Object Request Broker Architecture (CORBA). [BUTLER96] defines two groups of services: infrastructure services, and the services provided on behalf of common support applications. Among the former are communications services such as the TCP/IP applications Simple Mail Transport Protocol (SMTP), File Transfer Protocol (FTP), and Telnet, security services, name services, time services, object interchange services, data management and file management services, and presentation services such as print and device services. Common support application services include alert services, correlation services, message services, and MCG&I services.

Perhaps the most distinguishing factor of a server is the type of data, or data category, for which it is responsible. Rather than being categorized by the source, data is categorized by what it describes. In grouping data by what it describes, more efficient algorithms can be developed for parsing, sorting, and archiving that data, and more stable Application Programming Interfaces (APIs) can be written to request and process that data.

D. JOINT MARITIME COMMAND INFORMATION SYSTEM (JMCIS) '98

JMCIS is the maritime variant of GCCS, and JMCIS'98 is the latest version of the system. JMCIS'98 marks a radical departure from the legacy systems from which it descended. A principle tenet of the JMCIS'98 project is to significantly leverage COTS systems and the Windows/PC architecture.
As JMCIS is the target maritime C4I Workstation, it is incumbent upon the Marine Corps to be active agents in the process of identifying CAC2S requirements as they pertain to the JMCIS architecture and to ensure that CAC2S components are fully capable of seamlessly integrating within that architecture. JMCIS component programs, such as CAC2S, are responsible for documenting, validating, and presenting Operational Requirements Documents (ORDs) to the Copernicus Requirements Working Group (CRWG), the semi-annual forum for soliciting and prioritizing JMCIS requirements from the maritime services. A CRWG database with a Web-based interface provides the mechanism for JMCIS component program managers to track the progress of their requirements from identification to deployment. The database is accessible, on the World Wide Web, at http://copernicus.bahsd.com.

A keystone document, [JMCIS97], elaborates on the planned migration of JMCIS from a network of heterogeneous UNIX systems to a Web and PC-based operating environment, leveraging the private sector investments in Information Technology (IT) and simplifying maintenance and training on IT systems. Specifically, [JMCIS97] identifies six key tenets of JMCIS'98: migration to the DII COE, migration to PC Workstations and Servers, industry capitalization, combination of tactical and non-tactical networks, employment of "leading-edge" logistics, and streamlining of the acquisition process. JMCIS'98 will exercise an accelerated test/evaluate/certify/deploy cycle using the USS Coronado (AGF11) as a "Joint Battle Lab" to ensure suitability of the JMCIS'98 architecture and its components. Five architectural phases have been defined to enable the
migration of JMCIS to the target environment while concurrently meeting the operational requirements of the Fleet. The phases are identified under the JMCIS Unified Migration Plan (JUMP).

Finally, the Joint Maritime Communications System (JMCOMS) is the communications infrastructure upon which JMCIS'98 will be implemented. This infrastructure will be composed of a combination of high-speed LANs, lower-speed WANs, dedicated wireless SATCOM and LOS transmission, and on-demand (dial-up) service.

E. SUMMARY

The paradigm for performing requirements analysis and system specification for a complex, mission-critical, operational domain has clearly shifted. While the need to accurately identify domain-specific data elements and data processing functionality remains as critical as ever, now this analysis and specification must take place within the context of the architectural guidelines developed by the DII COE, GCCS, and JMCIS programs. A successful CAC2S operating environment will maximize the use of underlying infrastructure and common support application services, be designed to scale equally well across high-speed LAN and lower-speed WAN networks, and will create and deploy data type objects which facilitate their processing and transportation in a distributed TCP/IP network.
A. CHAPTER OVERVIEW

One of the keys to successfully migrating the MACCS from the suite of legacy equipment and processes it currently possesses to a DII COE compliant distributed computing environment is to identify the potential objects within the system. Those objects represent the data types manipulated by the system as well as the functional processes which provide the services to manipulate those data types. Objects are collections of data, or attributes, and the operations, or methods, which act on those collections of data. The design of a DII COE compliant MACCS cannot begin without a thorough and accurate specification of every object expected to occur in the MACCS domain. Every data type, from a track object to an imagery object, must be specified as a discrete collection of attributes: defined fields, values, and constraints, and a discrete collection of methods. Those methods are defined operations which permit the retrieval, initialization, and modification of those attributes. Every functional service, from track services to security services, must also be specified as an object. A track server, for example, would be specified as an object with index, communication, and storage components implementing the functionality of an abstract machine. That abstract machine would be capable of collecting, organizing, and distributing information on tracks, themselves abstract representations of real-world planned or active events. The importance of deliberate and exhaustive specification of the data types and processes which exist and act in the MACCS domain, and specifically, the treatment of these entities...
as objects which resemble and behave in predictable, real-world manner, is to take advantage of a future distributed computing architecture which will facilitate the movement and interaction of objects across a transparent, yet heterogeneous, network of dissimilar computing platforms.

Standards such as Distributed Computing Environment (DCE) and Common Object Request Broker Architecture (CORBA), protocols such as Hyper Text Transfer Protocol (HTTP) and Internet Inter-ORB Protocol (IIOP), and runtime environment paradigms such as the Java Virtual Machine (JVM) are currently being implemented inside and outside of the DOD and represent the distributed computing architecture of tomorrow. The following discussion of SAAWC data types and functional processes represents a beginning for the specification of the objects required to implement the desired SAAWC functionality.

B. SAAWC PROCESSES

[SAAWC95] identifies seven "displays", or user interfaces, required to provided the Sector Antiair Warfare Coordinator (SAWC) and the SAAWC staff with the information they need to conduct Air Defense Battle Management (ADBM). The SAWC is the chief architect of the ADBM plan and coordinates its implementation. The SAAWC is the collection of systems, personnel, and procedures used to execute the ADBM. These seven user interfaces can be used as metaphors for higher-level, or composite, "processes" within the SAAWC that can be further decomposed into client processes, broker processes, and server processes. Furthermore, this decomposition will identify processes,
such as a track serving process, which several upper level processes commonly require. The identification of user interfaces as "processes" has the additional advantage of providing for efficient scaling with minimal redesign effort. In a small-scale implementation of the SAAWC, for example, a single workstation or processor may interleave all of the SAAWC processes, permitting the operation of all the SAAWC user interfaces on a single machine. Alternatively, in a large-scale implementation of the SAAWC, a single workstation or processor might be dedicated to each process, providing not only one machine for each of the SAAWC user interfaces but also one machine each dedicated to such processes as a data management server, communications server, track server, and so on. Either small-scale or large-scale implementations could include spare workstations or processors loaded with client and server processes to provide the system with a robust, fault-tolerant design. A complete representation of all SAAWC functional processes is presented as figure 1.

SAAWC functionality is described in [SAAWC95] as requiring the following user interfaces: Air Defense Mission Display (ADMD), Offensive Air Support (OAS) Mission Display (OMD), Air Defense Situation Display (ADSD), Communication Status Display (CSD), Equipment Status Display (ESD), Intelligence Display (ID), Air Situation Display (ASD). Together these user interfaces enable the SAWC and his staff to maintain a timely, accurate BA, plan future air defense missions and postures, and identify air defense requirements which aren't being met. In addition to these missions, the SAAWC must also
Figure 1. SAAWC Functional Processes
be capable of becoming the alternate Tactical Air Command Center (TACC), should that facility become a casualty.

The first of the SAAWC user interfaces, the ADMD, will contain information from the Air Tasking Order (ATO) relevant to the prosecution of air defense. Within those subsections of the ATO deemed pertinent, the display must include information regarding mission numbers, call signs, mission type, ordnance/fuel, time on and off station, IFF codes, mission status, terminal control agency and frequency, package designator, routing information, and tanker availability.

The OMD will contain information from that subsection of the ATO pertaining to Close Air Support (CAS), Deep Air Support (DAS), Air Reconnaissance, Electronic Warfare (EW), and Offensive Antiair Warfare (OAAW) missions. The required information within those subsections of the ATO is identical to that of the ADMD, with the exception that more detail regarding air-to-air and air-to-ground ordnance must be provided.

The ADSD will present a real-time depiction of the current air battlefield picture. It must accomplish this through the presentation of "tracks", which represent air, ground, and seaborne entities which may have friendly, foe, unidentified, fixed, and mobile characteristics. Additionally, this display will include air defense warning conditions and weapons control status, Combat Air Patrol (CAP)/Fighter Engagement Zone (FEZ) manning, Missile Engagement Zone (MEZ) status as part of the Ground based Air
Defense (GBAD) status, States of Alert (SOA), missile inventories for ground-to-air missiles, and tanker assets, including tanker fuel availability.

The Air Situation Display will present a superset of the information presented by the Air Defense Situation Display with the purpose of enabling the Sector Antiair Warfare Coordinator and his staff to make timely decisions regarding future employment and deployment of air defense assets. The Communication Status Display will present to the SAWC and his staff a graphic depiction of communications personnel, equipment, and circuit status. Pertinent information will include circuit names, designators, equipment types, cryptographic means, frequencies, and status. The Equipment Status Display will present a battlefield picture of the operational status of equipment associated with designated higher, adjacent, and supporting units. Pertinent applications might include surveying the status of organic radar units, airfield control units, and weather forecasting units. The Intelligence Display will present a battlefield picture consisting primarily of static information. Friendly Order of Battle (FOB) and Enemy Order of Battle (EOB), designated facilities, friendly scheme of maneuver, and predicted enemy schemes of maneuver will be available to the Intelligence Analyst interacting with the ID.

C. **SAWC DATA TYPES**

There are a number of fundamental data types which enable the SAWC and his staff to plan, decide, and execute their assigned tasks. Foremost among these is the concept of a "track" data type. The conventional representation of a track is as a set of ascii text characters describing a real-world object, either fixed or mobile, ground-based,
air-based, or sea-based, and friendly, foe, or unidentified. It is the principal data type used to convey meaningful information to the MACCS operators and analysts regarding the prosecution of an air battle. At the most basic level, and as it exists in a Tactical Data Link (TADIL) system like TADIL J, a track is a bit-oriented subset of a data stream which is machine readable, and is manipulated by a special purpose TADIL processor to produce something which is man-readable and conveys meaningful information to an operator.

A TADIL J message is composed of, normally, 1, 2, or 3 Link-16 words, each word containing 70 bits of data. TADIL J, or Link 16, is the Joint Services and NATO forces standard for the exchange of real-time tactical data among units of the force. A J-series message has the potential for carrying information representing a real-world object's identification, status, activity, location, speed, bearing, and electronic operating parameters, along with information pertaining to the track itself, such as reporting source, track number, and track quality. The TADIL J implementation of the track data type is a significant enhancement to the process of forming a meaningful air battlefield picture.

Another fundamental data type is the character-oriented, man and machine readable, USMTF record message. The USMTF message set is a collection of mission and situation specific formatted message templates used to document and disseminate meaningful information on every subject from plans and operational orders to chemical attack reports and intelligence summaries. Of particular interest to the SAWC and his staff are the ATO and Situation Reports (SITREP). The ATO is the principal means for disseminating information pertaining to the conduct of aircraft missions (sorties) during a
given period of time. It is composed of subsets of information which describe the times, activities, units, platforms, payload, and communications details for specific air missions.

The ATO itself is a poor choice for a data type. Most units, and the SAAWC is no exception, are concerned with only that portion of the ATO which pertains to the prosecution of their assigned tasks. The ATO may still have utility as a tool for an ACE commander and his staff to review the projected employment of air assets during a given period of time. However, given the advances in messaging and database technology, the ATO is more appropriately considered as a user-defined collection of air missions which have been developed, documented, and disseminated for the purpose of setting up the air-related battlefield picture for a user-defined period of time. A single "air sortie" data type is a much more precise, efficient, and meaningful way in which to represent a real-world concept which, when executed, becomes the real-world object that is an active air sortie. In addition to providing a clear, simple transition to a track data type, an air sortie data type is much simpler to implement in a messaging and database driven environment.

The USMTF formatted SITREP message is a clear, succinct vehicle for conveying meaningful information pertaining to a unit's current status. As such, it is a good data type for representing the unit, equipment, and weapons system conditions and status that are required by the SAWC and his staff. The well-defined format and widespread adoption of the SITREP make it a logical choice for a data type capable of representing real-world
concepts: current operating status, current weapons alert conditions, current unit preparations. A SITREP data type is well-suited for both periodic and ad hoc messaging, and for database archival and retrieval.

A number of additional USMTF message formats are defined for the purpose of conveying special purpose information of central and peripheral interest to the SAWC and his staff. Requests for supplies, changes to unit employment, and task-related liaison with designated units are a few of the many needs fulfilled by USMTF messages. These real-world concepts should be examined in another forum as potential data types whose graphical representation could be location or functional area-based and whose representation as a data type serves the purpose of logging, alerting, and managing units, plans, and events as they occur during the prosecution of a battle.

D. PRODUCERS

The producers in a three-tier, DII COE compliant implementation of the SAAWC are those entities capable of generating computational representations of the real-world objects, concepts, and states which portray the battlefield picture at given intervals of time. They are those processes which, either through user input or electro-mechanical detection and identification, construct ADTs which can be transmitted by computer networks and manipulated by computer clients to form a meaningful depiction of a state of battle. Referring to [BUTLER96], a collection of producers would potentially include those common service providers representing the bottom tier of the client/broker/server architecture.
These common service providers include a track server, which would produce track objects representing real-world battlefield entities. Emphasis for the operation of a track server would be on near-real-time reporting of entity characteristics at the expense of possible duplicate, non-correlated, and incompletely identified tracks. The majority of tracks would be perishable in nature, and much of the value provided would be in their timely presentation to the weapons control, sensor control, or display client consumer. A secondary consideration for the consumption of tracks would be for their use in reconstructing events at a later date for the purposes of investigation or training. A track server might be implemented as a dynamic, real-time Relational Database Management System (RDBMS), in which the timing constraints facilitated by priority-based scheduling must be considered in addition to the traditional goal of guaranteeing database consistency.

A Data Management Server, described in [BUTLER96], would consist of three server types: an object-base server, a data management server, and a file server. An object server would produce "intelligent" objects, binary blobs of code which could facilitate communication between client objects and server objects. This data server would provide all the intelligence, i.e., knowledge of the attributes of objects and of their relations to other objects, that a client would need to establish a run-time liaison with a server object. The data management server would produce the interfaces required for communication between clients and various proprietary implementations of Relational Database Management Systems (RDBMS). The file server would produce a single name
space, hierarchical view of the file system in which all files are presented to the client as if they were located in a single tree structure, on a single system. Resolution of logical file names to their actual names and locations is accomplished by the Name Server, described below. The File Server, in conjunction with the Name Server, hides the actual physical location of files from the client, permitting the client to make requests for files in a manner most appropriate to its implementation. Additionally, the File Server enforces rules for the concurrent reading/reading and reading/writing of files by distributed clients as well as the required notification/flushing of invalid copies of files held by clients.

The Data Management Server is of particular interest to the SAAWC clients, as it would produce track objects representing evaluated and validated real-world objects which add intelligence to the battlefield picture. Known airfields, Surface to Air Missile (SAM) sites, and Early Warning (EW) radar sites are examples of static entities which might be served in a non-real-time manner to a weapons control, sensor control, or display client consumer. The implementation of a Data Management Server which provided static data in a track data type format would facilitate the transmission and reception of this data in a TADIL broadcast, as well as provide for the consumption of the data by clients generating request over traditional Local Area and Wide Area Networks. Finally, the implementation of a Correlation Server would be simplified by the specification of both perishable and non-perishable formats for a track data type by eliminating the requirement for translation between unlike data representations such as tracks and database records.

A Correlation Server would be a source for data fusion within the system,
providing "value-added" intelligence to the tracks it was subscribing to as a track consumer, producing a track with a higher level of assurance to subscribing consumers. Emphasizing data consistency and employing a set of business rules which enable the correlation of perishable tracks with one another, and perishable tracks with database records, the Correlation Server might generate tracks carrying a validity qualification which permitted their overwriting of less valid tracks stored locally at the subscribing consumer. The quality of a Correlation Server's produced tracks would be valuable both to a non-real-time track consumer and to a real-time track consumer, tasked both with providing what is known in the Intelligence community as data of an Indications and Warning nature as well as with providing data of an Intelligence (evaluated information) nature.

A Security Server would be responsible for providing the security services required to enable distributed processing and communication services which preserved the authenticity, secrecy, and integrity of the transmitted data. Specifically, the Security Server might produce certificates upon request by a client which enables the process of authenticating communications between the client and a specified server. The Security Server might also produce cryptographic keys which enable the client and server to communicate in a manner which disguises the nature of their communications. Finally, a Security Server might produce digital signatures for communicating processes which provides a mechanism for detecting the alteration of data enroute from a sending process to a receiving process.
The Security Server plays a critical role in a distributed computing environment in which mobile code transits not only LANs but also, increasingly, WANs. Centralizing the role of security in a server enables turnkey solutions, such as the Kerberos Server, to be implemented, while at the same time allowing intelligence to be built into clients permitting dynamic determination of the needs and levels of secure communication. Appendix C provides elaborating information on the Kerberos Server.

A Time Server simplifies the process of synchronization between networked processes. Each server in the three-tier architecture might subscribe to the Time Server, consuming the produced timestamps which facilitate synchronization among cooperating processes. Timestamps are a critical means for permitting correctness in computation and for auditing computing histories.

An Alert Server would produce simple messages, alerts, in response to an alert filter profile created by a subscribing client process. These messages could be in audio, visual, text-based, or graphics-based format, and they might reference the track, record, or message data type identified in the alert filter profile. An intelligent Alert Server implementation would subscribe to and monitor both real-time and non-real-time streams of data types transiting the network, and generate alerts which contained references to the identification and location within the system of the subject data type. Centralizing alert services permits the reduction of redundancy within the system. If, for example, several client consumers have indicated interest in the release of a pending operation order message, only one process need monitor the network for that message, and only one copy
of that message need be saved and referenced by the alerted clients. An Alert Server should be capable of producing alerts based upon profiled real-world objects, real-world concepts, and real-world events, as well as computer-related conceptions such as alerts to other servers, for example, to initiate back-ups to their secondary or off-site storage.

A Map Server would produce the appropriate vector or raster depiction of the requested display background. This server could function as a single-point repository for navigational charts, tactical maps, and digital images, as well as vector-based geographical displays. Additionally, non-traditional maps, such as the inside of a warehouse, or a computer-driven electronic "message board", could be produced, rendered as vector diagrams, and used by subscribing clients to provide spatial and chronological context for supply "tracks" or record message "tracks" which are graphically displayed on the client machine. The primary responsibility of the Map Server is to produce background displays which provide the most appropriate context in which the user can draw conclusions from the overlaid "track" information. Whereas initial implementations of a Map Server might force the client to subscribe to specific display types and scales, an intelligent implementation might make display type and scale decisions based upon the "clutter" of tracks, display resolution, display size of the consuming client, and user selection.

A Name Server would provide the critical function of mapping logical names to physical locations throughout the network. The Name Server would produce references to the real-world objects, concepts, and events in the naming context which was most appropriate to the subscribing consumer, resolving the requests to determine the correct
identification and location of the named entity. The resolved names produced must permit quick, efficient, and accurate communication across the network between the subscribing client and the server storing the resolved entity. A single source, name-resolving process simplifies the operations of adding and removing clients, brokers, servers, devices, and network protocols to and from the network.

A Message Server would be a client subscriber to the Communications Server, parsing, archiving, and forwarding to subscribing clients references to requested messages. The intelligence built into a Message Server would consist of the functionality to recognize message formats and to properly distribute messages to the primary or secondary storage of its subscribers. For example, tracks from a non-TADIL source would be forwarded to the Track and Correlation Server for processing and archival, while SITREPS would be forwarded to a local message library and to addressed subscribers. A Message Server is a producer in the sense that it takes character-based man-readable and machine-readable data from the Communications Server and produces a formatted message recognizable to subscribing clients. A Message Server might also be a repository for message format templates which facilitate the correct drafting of messages by clients, as well as a repository for message addressee templates which simplifies the process of addressing messages.

Finally, a Communications Server would manage the interfaces required by clients to access and employ application level processes specific to the underlying network protocols. For example, the Communications Server would provide TCP/IP-based
SMTP, SNMP, FTP, Telnet, and HTTP services to subscribing clients, enabling them to be designed in such a manner as to focus on data type and service type needs rather than on communication type specifications. A Communications Server would work hand in hand with a Network Server, translating messages into data streams packaged for communication over particular datalink, network, and transport layer configurations, in addition to unpacking received data streams for subscribing clients.

E. CONSUMERS

In the truest sense of the term client, most producers are themselves also clients of certain data types and processes. For example, every producer would subscribe to the broker for timestamps provided by the Time Server. Certainly, our broker in the three-tier architecture is also a client of the data provided by the various servers. However, for the purposes of this paper, the clients of interest are those identified in a review of required functionality for the SAAWC: Air Defense Mission Display, Air Defense Situation Display, Offensive Air Support Mission Display, Air Situation Display, Intelligence Display, Communications Status Display, and Equipment Status Display.

These clients are themselves composite entities containing atomic clients: they are the principal interface between the human operator and the processes which receive, store, manipulate, and present data in a meaningful manner to him. It is in this context that clients are described. They are presented as the composite operator which organizes and coordinates the processes responsible for consuming requested data elements as well as commands from the human operator.
The ADMD client and the OMD client both exist for the purpose of presenting graphical representations of those portions of the ATO pertinent to the respective Displays. This is the principal means by which the human operator is able to develop a comprehensive FOB for a given period of time, for the purpose of prosecuting air defense and offensive air support tasks. The two display clients consume the digitized charts, maps and images, received from the Map Server, and present them to the operator in a manner which conveys the spatial and chronologic ordering of air sorties described in the ATO. The two display clients also consume queried air sortie data types, presenting them as tracks representing the concept of planned but not yet executed missions.

By focusing on the background display needs of the human operator, and using the inherent geographic, functional, and chronologic attributes of the planned air sorties, powerful relations can be drawn and displayed to the operator which relate the planned air battlespace in a manner which provides intelligence to the battlefield. For example, if an operator desires to see a given area of the air battlefield, then his request should be interpreted not only as an invocation to display the geographic area, but also as a query into the sortie database to display those air sortie tracks which are related to the requested area. In another example, if an operator desires to see chronologic coverage of a needed asset, say air refueling, then a request to display a timeline should also invoke a query to display those air sortie tracks functionally related to the request, and in a manner which conveys their relation to the displayed timeline. By considering, designing, and creating these different data elements, charts, timelines, and planned sorties, as
computer-generated, object-oriented manifestations of real-world concepts, we can
develop computer-derived presentations of real-world events which encapsulate the
meaning of activities such as personnel and equipment moving through the air battlefield.

Furthermore, by describing data elements, such as air sorties, as tracks
representing unexecuted missions, it provides for a simple, yet powerful means by which
real-world concepts can be transformed into the real-world events which are of interest to
the ASD, ADSD, and ID clients. These clients are the principal means by which the
SAWC and the SAAWC operators monitor the real-time events of the air battlefield.
These clients also should incorporate the mechanisms for consuming and presenting to the
operator background displays which convey the geographical and chronologic boundaries
of the air battlespace, as well as the activity within those boundaries, on behalf of the
respective air and air defense controllers. The consumed tracks, in this case, are those
tracks representing actual missions, sea-based and ground-based as well as air-based,
which have a direct bearing on the controlled battlespace. All tracks would consist of
geographic, functional, and chronologic attributes which mark them for retrieval from a
locally stored, dynamically updated database, to be displayed upon the triggering
background display.

The radical departure from the traditional means of depicting the battlespace as a
two-dimensional field of sensor-derived contacts, as is the case with current TADIL
processor and display subsystems, would be a shift in recognition that the source of data is
much less important to the operators and analysts than is the type of that data. In a battle

38
environment where tactical events begin as concepts, become executing events, influence other executing events, then evolve into historical occurrences, the common thread that is the grouping of personnel, equipment, and tactics should be encapsulated into a common, consistent data type. That data type would be composed of attributes which illustrate its state, facilitate its manipulation by computing processes, and permit its representation to a human operator in ways which allow him to draw intelligent and meaningful conclusions about the prosecution of battle.

The CSD and the ESD clients principally consume information regarding the operation of the components responsible for the SAAWC communication, computing, and presentation infrastructure. A traditional interpretation of this functionality consists primarily of equipment and network status messages which indicate the current operating condition of the system devices and communication links. A broader interpretation includes status messages indicating traffic flow, logging of system events, and information pertaining to the configuration of the equipment such as the procedures traditionally found in the Communications Annex, Annex K, of an operational plan. The CSD and ESD clients might consume geographical displays from a Map Server for presentation to the user as a background, representing an abstraction of the real-world locations and activities of his communications and computing devices. Status messages might be objects, which are generated and modified by Communications and Device Servers, with methods permitting the display of the objects as tracks, or their forwarding to other clients for action. What is of significant interest to these clients is that these configuration and status
messages be modeled as objects which represent the abstract notion of the state of computing machines and computing networks, and are defined in a manner both to make their manipulation and presentation by clients responsible for these activities simpler, and to facilitate this presentation so that it contributes to the notion of a COP and provides intelligence to an otherwise dissimilar and unrelated collection of events.

F. SUMMARY

The preceding discussion of data types, processes, producers and consumers, which are all part of the SAAWC operational domain, is intended to illuminate the process of developing computing domain implementations of operating domain entities. This is the critical first step in the process of designing an Automated Information System (AIS) solution to an identified operational need: in this case the need to build a general purpose, "Battlefield Awareness Machine", capable of bringing coherence to the air battlespace for which SAAWC operators are responsible.

An important observation, regarding the processes and services described above, is that many are generic services typical of those found in a distributed computing environment. They are also common to the command, control, communications, intelligence, administration, logistics, and other assorted activities taking place throughout a given organization at the tactical, operational, and strategic levels. Furthermore, many, if not all, of these services are currently specified and implemented both in Commercial Off The Shelf (COTS) and Government Off The Shelf (GOTS) solutions which are part of current, or planned, versions of the GCCS and Global Combat Support System (GCSS).
IV. PROTOTYPE DESIGN

A. CHAPTER OVERVIEW

This chapter describes the methodology used to develop the SAAWC prototype with the CAPS tool, as well as the steps taken to build that prototype. Also described are the various tools used in addition to CAPS to model process interaction and message flow in order to produce a prototype which most accurately captures the behavior exhibited in the SAAWC. The chapter serves to document the analytical process responsible for the composite and atomic functionality designed into the SAAWC prototype.

B. COMPUTER AIDED PROTOTYPING SYSTEM (CAPS)

The Computer Aided Prototyping System is a software engineering solution to the need to rapidly develop executable prototypes from user specifications to contribute to the accurate, correct, and satisfactory development of Automated Information Systems. CAPS was developed at the Naval Postgraduate School as a tool for modeling real-time embedded software systems which exhibit the control and timing constraints expected in the modeled system itself. CAPS is an operating environment which consists of a set of tools which permit the prototype designer to graphically depict functional operators and data streams at composite and atomic levels. CAPS uses a fifth-generation prototyping language, PSDL, which provides the designer with the mechanisms needed to implement the timing and control constraints, messaging, and data typing in the prototype. Through an integrated scheduler and translator, CAPS creates the Ada language specifications for each atomic operator, data stream, and user-defined data type. In addition, CAPS
provides the implementation for a main procedure which dictates the timed behavior of the prototype and how exceptions developed during prototype runtime are handled. CAPS includes the functionality to invoke an Ada compiler for the production of the prototype executable code.

C. PRELIMINARY DESIGNS

In preparation for using the CAPS graphical editor to determine composite and atomic operators and associated data streams, preliminary designs were constructed using the X-Windows drawing tool known as xfig. This approach proved beneficial for two reasons: operator and data stream placement could more easily be initiated and changed, in attempts to reduce screen clutter and improve viewing clarity, because the screen redrawing was more efficient; and additions and deletions to the existing operators and data streams were not associated with the creation and modification of PSDL code, which lessened the likelihood of garbage declarations occurring in the PSDL code during the CAPS graphical editing phase. Drawings with the xfig tool were created to represent the high-level SAAWC as well as single-level decompositions of all the SAAWC composite operators. A sample drawing, of the decomposed Broker operator, is presented in figure 2.

A second tool, Microsoft Excel spreadsheet, was used following the stabilization of the xfig drawings to represent event sequences corresponding to desired prototype behavior for typical scenarios. A spreadsheet was created with the horizontal axis representing SAAWC composite operators and the vertical axis marked by consecutive
Figure 2. Broker Operator Drawn with Xfig
labels for points in time during prototype execution, appearing in chronological order. Spreadsheet cells were populated with a text value representing two concepts: the first is the data stream label associated with the originating SAAWC composite or atomic operator, and the second is a label which uniquely identifies the event which triggered a series of related messages. For example, in figure 3, row 3, column 3, the cell value "ad_sub_tracks=tracks_I_want" represents a message sent from an atomic operator within the composite operator Air Defense System Display indicating a request to subscribe and receive notification from the system of the presence of a track, or set of tracks, with the particular request being associated with the tag "tracks_I_want." The spreadsheet was edited to depict at least one series of messages originating, terminating, or passing through every SAAWC composite operator. The complete spreadsheet is included as Appendix B.

This presentation of associated data flow triggered by system events facilitated the determination of accuracy, correctness, and thoroughness of the initial SAAWC diagrams. In one example where the spreadsheet highlighted an initial system design flaw, during the process of tracing message flow which resulted from a request to display track information, it was determined that such a request logically should also trigger subscriptions and requests to the Correlation Server and to the Data Management Server as well as to the more obvious Track Server.
<table>
<thead>
<tr>
<th>TIME</th>
<th>MASTER</th>
<th>AIR DEF SYS DISPLAY</th>
<th>BROKER</th>
<th>CORRELATION</th>
<th>DATA MANAGEMENT</th>
<th>TIME</th>
<th>TRACK</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>now_Enum755190500821</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>filter_tracks_tracks, I want</td>
<td></td>
<td>filter_tracks_tracks, I want</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>ad_sub_tracks_tracks, I want</td>
<td></td>
<td>ad_sub_tracks_tracks, I want</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>client req_tracks, I want</td>
<td></td>
<td>client req_tracks, I want</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>valid client req_tracks, I want</td>
<td></td>
<td>valid client req_tracks, I want</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>req_tracks_tracks, I want</td>
<td></td>
<td>req_tracks_tracks, I want</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>request_tracks_tracks, I want</td>
<td></td>
<td>request_tracks_tracks, I want</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>filter_tracks, I want</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>respond_tracks_tracks, I want</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>filter_tracks, I want</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>now_Enum755190500821</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>subscribe_tracks_tracks, I want</td>
<td></td>
<td>subscribe_tracks_tracks, I want</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>subscribe_db_changes_tracks, I want</td>
<td></td>
<td>subscribe_db_changes_tracks, I want</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>ca_dal_tracks_tracks, I want</td>
<td></td>
<td>ca_dal_tracks_tracks, I want</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>track_db_tracks, I want</td>
<td></td>
<td>track_db_tracks, I want</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>req_data_tracks, I want</td>
<td></td>
<td>req_data_tracks, I want</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>req_data_type_tracks, I want</td>
<td></td>
<td>req_data_type_tracks, I want</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>req_ret reluctance_tracks, I want</td>
<td></td>
<td>req_ret reluctance_tracks, I want</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>req_ret msg_tracks, I want</td>
<td></td>
<td>req_ret msg_tracks, I want</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>respond_msg_tracks, I want</td>
<td></td>
<td>respond_msg_tracks, I want</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>now_Enum755190500821</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>respond_db_change_tracks, I want</td>
<td></td>
<td>respond_db_change_tracks, I want</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>record_db_tracks, I want</td>
<td></td>
<td>record_db_tracks, I want</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>valid_tracks_tracks, I want</td>
<td></td>
<td>valid_tracks_tracks, I want</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>25</td>
<td>respond valid_tracks_tracks, I want</td>
<td></td>
<td>respond valid_tracks_tracks, I want</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>26</td>
<td>manage_templateadd_spot_rep tmp</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>27</td>
<td>srv_response_tracks, I want</td>
<td></td>
<td>srv_response_tracks, I want</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>28</td>
<td>valid srv_response_tracks, I want</td>
<td></td>
<td>valid srv_response_tracks, I want</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>29</td>
<td>ad_del_tracks_tracks, I want</td>
<td></td>
<td>ad_del_tracks_tracks, I want</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
D. CAPS GRAPHICAL EDITING

1. High-level Prototype Decomposition

The purpose of the CAPS SAAWC prototype, presented as figure 4, is to model the data flow and functional operator interaction which exists within the modeled system, a SAAWC application segment running in the JMCIS environment. In the design of the prototype a decision was made to narrow the scope of simulated SAAWC behavior by including only a decomposition of that functionality specified for the Air Defense System Display (ADSD) operator. This decision was made after analysis of the modeled system indicated overlapping functionality between the seven SAAWC functional operators. The selection of the remaining operators for the high-level SAAWC prototype followed from the identification in previous work of the user interface and of the common infrastructure support and common support application functionality. Three operators were added to simulate external feeds of data or mechanisms for user interaction: a network operator, TADIL_J operator, and a get_user_command operator. An implementation of the network operator would simulate a LAN itself, presenting network-specific packages, in this implementation consisting of a user-defined data type called "bits", to the Communication Server composite operator for further processing. The network operator would similarly receive from the Communication Server operator messages which had been packaged as "bits", and were prepared for transmission onto the simulated network. An implementation of the TADIL_J operator would simulate a feed from a TADIL_J processor, itself capable of presenting "tracks" representing mobile operational entities,
Figure 4. CAPS SAAWC Prototype
packaged in a track file which is transmitted periodically. A final additional operator, get_user_cmd, would be implemented in a Graphical User Interface (GUI) programming tool, such as the TAE\(^1\) and would be the means by which the executing prototype interacted with the user utilizing the standard input and standard output streams for communication.

The logic behind the selection of particular data streams for implementation at the high-level SAAWC, to carry messages between the specified composite and atomic operators, resulted from the previous analysis of the functionality of the operators themselves. Examination of the ADSD functionality, for example, identified the need to accept from a user requests to process and archive "filters", or requests, representing tracks, messages, alerts, database records, and map displays pertinent to the prosecution of specific user tasks. Data streams for each of these messages, as well as data streams to carry composed messages, print requests, and requests for secure communications sessions, were identified and drawn to convey the information required for the user to elicit the desired behavior from the prototype. At this level of the ADSD operator, moreover, it is assumed that some sort of internal processing would take place. This processing might consist of validation of these message requests or of archival and modification of locally maintained filters prior to transmission of the messages, intact, to the Broker operator. This internal processing is elaborated upon in the section describing the decomposition of the ADSD composite operator.

\(^1\)TAE is a trademark of the National Aeronautic and Space Administration.
The Broker operator is the critical archiving, controlling, and validating component in the three-tier architecture. At the highest-level, the complexity of its decision-making is hidden. The operator was drawn to convey the transmission of messages, triggered upon receipt of messages from various originating operators, to the appropriate destination operators. In fact, all outgoing data streams were drawn at this level as responses to the associated incoming data streams. For example, an incoming request to subscribe to the Map Server generated a request to the Map Server, which in turn generated a message containing either the map location or the map itself, triggering a Broker message to the requester which would contain the map itself. One additional data stream was drawn to carry Broker configuration information from the user to the Broker. Implementation of this functionality would permit a user to directly influence the decision-making logic of the Broker, determining such things as priority of delivery of tracks, validation of service requests of particular clients, and discrimination of duplicate or time-late information from a server.

The Alert Server composite operator was drawn with two inputs: messages containing track information, and user-generated requests to set a filter for specific track information. A single data stream output, an alert, would be generated by a successful reconciliation between an internal track message database and a particular filter request. This reconciliation process, amplified by the Alert Server decomposition, would be triggered by the receipt of either a new track message or a new filter request message.

Similarly, the Track Server composite operator was drawn to indicate message
inputs from two sources: the TADIL_J operator and the Broker operator. Messages coming from the Broker represent user-generated requests for specific "tracks", and the messages coming from the TADIL_J operator represent the actual tracks themselves. The Track Server would encapsulate the behavior necessary to reconcile requests for tracks against an internally managed track repository, responding to successful matches with a corresponding message to the user.

The Communications Server composite operator was drawn to be the interface between the SAAWC itself and the external network. To this extent, the Communications Server operator is illustrated receiving messages from, and delivering messages to, the network operator. Additionally, data streams representing unpacked FTP, HTTP, SMTP, SNMP, and UDP messages were drawn from the Communications Server operator to the Broker operator, indicating some internal processing of the network generated messages prior to their routing to the appropriate destination. This processing might include extracting the TCP/IP application packets, verifying them for correctness, assembling complete messages, or perhaps validating timeliness of delivery. Finally, two data streams are drawn from the Broker operator to the Communications Server operator: the first, session_data, represents a SAAWC user-generated message destined for an addressee outside the SAAWC network, and the second, session_control, represents a SAAWC user-generated control message indicating a desire to set up a particular TCP/IP application session between the originator and a specified destination. For example, a user generating an HTTP request for the first time might in fact generate two messages, the
first telling the Communications Server to establish a "thread" of communication between the user and the local HTTP server, perhaps acting in the role of a "proxy" WWW server, while the second message would contain the contents of the page, or resource, request itself.

The Device Server and Map Server are composite operators drawn with highly refined, single-purpose functionality. A data stream from the Broker operator to the Device Server represents an administrator-generated message to add, modify, or perhaps delete a physical device resource to, or from, the network. The outgoing device_status message is both an indication to the administrator of the success of the device installation, as well as a trigger to the Name Server composite operator to modify its Name database so that user-generated requests to utilize a particular resource are properly mapped to the actual location of that physical device.

The Map Server composite operator is drawn in a similar manner, illustrating the generation of a map message, containing the map itself or perhaps a reference to a map file that is physically located elsewhere. This is triggered in response to an incoming message, request_map, representing a user-generated request to receive a particular display background for local manipulation.

The Correlation Server composite operator encapsulates functionality for providing "intelligence", or added value, to the tracks processed in the SAAWC. For this reason, the Correlation Server operator was drawn to illustrate "black box" processing of incoming tracks and filter track requests, just as the Alert Server and Track Server...
operators were. But the Correlation Server operator was also drawn to indicate an outgoing message subscription to the Data Management Server. A corresponding incoming data stream, respond_db_change, represents the Data Management Server's message response containing the requested database record information. It is presumed, at this level, that some unspecified internal processing exists to correlate track information with database record information and that successful matches will be delivered to the user, via the Broker operator, on the respond_valid_tracks data stream.

The Data Management Server is a composite operator providing functionality to query, modify, add, and delete database records in response to user requests. In order to simplify the interface presented to potential clients of the Data Management Server, the operator was drawn with a single incoming data stream, encapsulating the request attributes, such as database record type, index, and request type. It is presumed that some internal processing of the messages, to extract and route the requests to the designated atomic operators, is provided. A single corresponding outgoing data stream represents the success of the user request and, if appropriate, the results of the request.

The Message Server composite operator is envisioned to be an archival service for the management of incoming and SAAWC user-generated USMTF messages. Not unlike the Alert Server, Track Server, and Correlation Server, the Message Server operator is drawn to convey some internal processing of the incoming messages it has subscribed to receive, as well as the incoming message filter requests generated by the ADSD client. A corresponding outgoing data stream, respond_msgs, represents matches made between the
received and requested messages. Additional functionality has been incorporated into the Message Server, however, to provide for message template service to the ADSD client. It is presumed that an internally managed template database would be available to users requesting message templates, via the respond_template data stream, by serving up a formatted file, or a reference to a formatted file, with which the user could initiate the message-generation process.

The Name Server composite operator provides critical, single point, logical resource to physical location mapping. In the absence of such an implementation each client, including every server acting in a client capacity, would require local updating on every occasion that a new resource was added to the system. Such a scheme would quickly become unmanageable. Two incoming data streams, req_resource and mod_obj_loc respectively, correspond to client-generated requests for resources and to administrator-generated messages effecting changes to the locations of those resources. The lone outgoing data stream, resource_phys_loc, carries the information needed to effect communication with the resource itself, or, in the case of a file resource, initiate the file transfer process.

The Print Server composite operator was illustrated to reflect incoming print requests and corresponding outgoing print job spooled messages. Additionally, it was presumed that print requests might be one of two varieties: existing text, map, or graphic files, or screen grabs. To account for the former case, the data stream request_file was drawn to represent a message to the Name Server for file location information, and the
data stream receive_file drawn to represent the corresponding response. To account for
the latter case, some black box processing is presumed for the acquisition of ADSD
display parameters and the building of a file into the appropriate format for printing.

The Security Server is a composite operator encapsulating the functionality
necessary to provide secure communication sessions which preserve the secrecy, integrity,
and authenticity of the session content. Incoming data streams, representing
user-generated requests for certificates, keys, and signatures, were drawn to depict the
initiation of a secure session, on behalf of any client to any other client or server, with the
corresponding issue_key, issue_certificate, and issue_signature data streams drawn to
represent the Security Server operator's responses. Internal processing is presumed at this
point to include at least the generation, archival, and processing of all certificates, keys,
and signatures, required by the ADSD client and system servers in the course of
conducting communication sessions.

Finally, the Time Server operator is drawn as a composite operator with the
functionality to produce periodic timestamps, represented by the data stream, new_time,
which are then used by the Broker operator to orchestrate the message receipt, course of
action prosecution, and message-generation upon which the entire system relies for
efficient and predictable behavior. Decomposition of the Time Server operator is
expected to reveal the functionality required to develop and operate a global clock,
capable of generating accurate, fault-tolerant system timestamps.
2. Air Defense System Display (ADSD) Decomposition

The ADSD, presented in figure 5, was decomposed to incorporate the functionality necessary to retain locally managed state for user-defined track filters, database record filters, message filters, map display filters, and alert filters. The provision of atomic operators which maintain the state of a user's outstanding filter requests allows for efficiencies to be implemented at the ADSD level. For example, were a user to request track information on all F/A-18s in a particular air sector over a given 15 minute period of time, and then follow that up with a request for track information on all F/A-18s in the same air sector over a 30 minute period, rather than build a new filter to manage incoming tracks which meet this new description, the atomic operator responsible for managing track filters, track_display_db in the prototype, would reconcile the time fields of the original filter request and generate a subscription message intended to reflect this modification to the Broker and Track Server operators. Locally maintained state can also reduce message traffic within the system by validating user-generated requests denying, for example, requests which are redundant, incomplete, or unachievable within established time constraints. Each of the five filter request operators are drawn to indicate the receipt of filter modification messages, the self-generation of those messages back to the operators to indicate an effected change of state, and the generation of new subscription messages intended to indicate to the appropriate service a change in the desired service requests.

A security_manager atomic operator was developed to incorporate the
Figure 5. Air Defense System Display Operator
functionality required by a user desiring the initiation of a secure session with another client or server. The operator was drawn to illustrate the triggering of independent requests for the components of a secure session: certificates, keys, and signatures. These requests are based upon incoming messages stored in the msg_out data stream which are generated by the user developing content, for example, USMTF messages or e-mail, intended for transmission to a designated recipient. The content would be held pending arrival of the secure components, indicated by the iss_certificate, iss_key, and iss_signature data streams, whereupon the security_manager operator self-generates state modification messages, wraps the content accordingly, and places the resulting secure message on the outgoing data stream secure_msg_out. Efficiencies achieved by a locally maintained state operator, such as security_manager, include the reuse of existing certificate, key, and signature information during multiple secure sessions between the same parties, and the validation of secure session requests based upon predetermined rules for session requests with specific destination clients and servers.

A final ADSD functionality is illustrated with the print_manager atomic operator. A stateless operator, the print_manager would process print requests ensuring, for example, that the requests were properly formatted, addressed, and prioritized. The corresponding message, print_response, would be processed with information being displayed to the user indicating the result of the print request and, perhaps, amplifying information on the handling of the print request by specific printing devices.
3. Alert Server

The Alert Server, presented in figure 6, is decomposed into three atomic operators. The alerts.update-filter operator incorporates the functionality required to retain knowledge of all received alert requests. The state maintenance of this atomic operator is indicated by the self-generated state stream, alerts_filter, which is triggered by an incoming message, req_alerts. The alerts_filter state stream both modifies the internally managed database of alerts requests and carries the most recent alert filter request to the operator resolve_alerts, itself responsible for resolving existing alert filter requests with incoming messages. The atomic operator resolve_alerts receives incoming messages on the data stream alerts_message_db, which is triggered in response to the receipt at operator update_msgs of incoming messages generated by the composite operator's subscription to the Message Server. Like the alerts_update_filter, the update_msgs operator maintains state through the self-generated state stream, alerts_message_db, which modifies an internally managed database of incoming messages.

The critical resolution process, whereby alert messages are generated and put on the respond_alerts data stream, is encapsulated in the resolve_alerts operator which itself is triggered upon receipt of messages from either of the alerts_update_filter operator or the update_msgs operator. One note on the potential implementation of the alerts_update_filter operator: in order to preserve modularity and independence in the development of client operators, this operator should be completely unaware of which clients are requesting which particular alert filters. Instead, this operator maintains
Figure 6. Alert Server Operator
knowledge only of what messages are to be monitored for the generation of alerts. Such an implementation simplifies the record-keeping required of the alerts_update_filter and passes the burden of associating specific requests to particular clients to the Broker operator, where it must reside in order to allow development of an Alert Server which has no knowledge or dependence on potential client implementations.

4. Broker

The Broker operator, presented in figure 7, is decomposed into three atomic operators. The client_thread_manager operator processes requests from clients, or servers acting as clients, either constructing new threads, modifying existing threads, or destroying threads in response to those requests, and puts responses to those requests for service on the data stream client_req, as appropriate. This data stream is drawn as a state stream, indicating that the message on the data stream is used not only to trigger the business_rules_manager operator, but also to modify the state of the internally managed client thread database. The client_thread_manager also receives the data stream valid_srv_response, representing a response to a client request for service. It processes the content of this message, updating the record-keeping associated with that particular client request, and forwards the message on the relevant data stream corresponding to the initial request.

The business_rules_manager atomic operator is where the "personality" of the entire SAAWC system resides. This is the operator which contains the intelligence to adjudicate client-generated requests for service, validating not only client requests but also
server responses, prioritizing delivery of those responses to real-time and non-real-time clients alike, and correlating requests for specific types and values of information to reduce the number of duplicate transmissions throughout the network. The business_rules_manager operator processes the client_req data stream and responds to valid requests by generating the valid_client_req message to the server_thread_manager operator. Similarly, it processes the srv_response data stream and responds to valid server responses by generating the valid_srv_response message to the client_thread_manager.

The mod_rules data stream represents the interface between the system administrator and the system for effecting modifications to the set of rules for determining system-wide prioritization, validation, and domain-specific logic. The mod_rules data stream is represented as a state stream, indicating self-modification of the state of the business_rules_manager operator through the content of this data stream.

The server_thread_manager atomic operator functions much as the client_thread_manager operator does, maintaining a database of all current subscription threads to system servers, each thread referencing a capsule of data attributes which indicate the status, originator, destination, and message content associated with a thread session. The operator updates itself through the state stream srv_response, a copy of which is also passed to the business_rules_manager for content validation.

5. Communications Server

The Communications Server operator, presented in figure 8, is decomposed into two sets of TCP/IP application processor operators, two network interface operators for
Figure 8. Communications Server Operator
handling messages delivered to, and received from, the network, and a session_manager
atomic operator which manages the establishment and maintenance of TCP/IP application
sessions initiated by the ADSD client. The two sets of TCP/IP application processors
represent a subset of the larger group of TCP/IP application protocols in use today. The
Simple Mail Transfer Protocol (SMTP) provides for simple management of e-mail client
and server operations, the Hyper Text Transfer Protocol (HTTP) is a mechanism by which
World Wide Web pages are transferred from servers in response to client requests, the
Simple Network Management Protocol (SNMP) provides a means for the monitoring and
management of network operations, the User Datagram Protocol (UDP) is used for
connectionless data transfers in which dropped packets can be tolerated, and the File
Transfer Protocol (FTP) provides a mechanism and a set of commands for effecting the
transfer of files between nodes on the network.

Each of these protocols is represented by both an outgoing process operator and
an incoming process operator. The outgoing process operators take client-generated
messages and package them for transmission on the network. Protocol-specific message
segmenting, labeling, and prioritization are among the potentially implementable activities
at this level. The incoming process operators receive the respective protocol encapsulated
packets, strip the protocol header information from the messages, reconstitute the
messages into their original, pre-transmission state, and forward the messages to the
addressed destination.

The network interface atomic operators implement specific network datalink-layer
functionality, packing and unpacking, segmenting and reconstituting, transmitting and retransmitting packets as necessary. The characteristics of datalink-layer transmission protocols, such as IEEE 802.3 or IEEE 802.5, would be simulated at this level in order to evaluate potential performance bottlenecks or constraints within the network. Finally, the session_manager atomic operator would manage and archive session requests in response to the user-generated requests for new sessions arriving on the session_control data stream. This data stream is implemented as a state stream, modifying the internally managed database of TCP/IP application sessions. The incoming session_data data stream carries the content of the session-specific request which may, in the event the content references an FTP session, for example, contain the commands and arguments required to initiate a file transfer between client and server.

6. Correlation Server

The Correlation Server operator, presented in figure 9, is decomposed into five atomic operators. Three operators implement the functionality of receiving archival messages, and of triggering self-generated responses which modify the internally managed state of the operators, while simultaneously putting the responses on outgoing data streams destined for a correlation processor. The three operators identify functionality for the archiving of track requests, database records, and tracks developed from sensor-derived information distributed throughout the network. The first of the two correlating operators, correlate_tracks, reconciles incoming tracks with database records in order to produce a "value-added" track with some higher degree of assurance as to the
Figure 9. Correlation Server Operator
track's authenticity. The second correlating operator, correl_handle_query, reconciles the value-added tracks with user-generated track requests, responding only with matches which represent the identification of a particular track, or set of tracks, of interest to the user. Finally, and as previously described, the two data streams, subscribe_tracks and subscribe_db_changes, represent requests from the Correlation Server to subscribe to the Track Server and Data Management Server in order to continually receive new track and database record information, reflecting both the dynamic nature of the rapidly changing battlefield and the changing needs of the ADSD operator.

7. Data Management Server

The Data Management Server, presented in figure 10, incorporates the functionality to determine the data type and request type of a particular data management request before passing the message content to a particular DBMS for action. Three DBMS types, an object DBMS, a conventional record DBMS, and a file DBMS, are specified in the Data Management Server as a collection of independent processes acting on requests for querying, modifying, adding, and deleting the request contents from the respective DBMSs'. Each DBMS consists of four similar atomic operators, each of which is triggered by a message received from the resolve_request_type atomic operator. With the exception of those atomic operators handling DBMS queries, or record retrievals, each of the DBMS atomic operators generates a state stream, for the purpose of self-modification, as well as generating an outgoing data stream, corresponding to the triggering input stream, which carries the result of the DBMS request back to the request
Figure 10. Data Management Server Operator
originator. All responses to DBMS requests are passed from the DBMS atomic operators to the format_response operator which packages the response in a DBMS-independent format for further transmission to the request originator.

The resolve_data_type atomic operator is illustrated as a process which receives the initial requests for DBMS action, determines the appropriate DBMS for service, and forwards the resolved request to the resolve_request_type atomic operator. An implementation of the resolve_request_type atomic operator would be made aware of the available DBMS methods through contact with each DBMS's published interface and would be responsible for invoking the correct method based upon the resolved request type indicated in the incoming req_data_type data stream.

A sample transaction, to delete a conventional database record from a database, would proceed as follows: the request would be identified as carrying data type database record in the resolve_data_type atomic operator; the request would be identified as request type delete database record in the resolve_request_type atomic operator; the delete database record request would be forwarded to the delete_record atomic operator where the record deletion would be effected, and a corresponding request status message would be generated; the request status message would be processed by the format_response atomic operator where it would be stripped of any DBMS-specific packaging and formatted for transmission to, and processing by, the request originator.

8. Device Server

The Device Server, presented in figure 11, is decomposed into a single, simple
atomic operator, `mount_device`. Triggered by the incoming data stream, `administer_device`, an implementation of the `mount_device` operator would retrieve the configuration file referenced in the `administer_device` message and apply the appropriate device driver information to enable the device on the network. Alternatively, the `administer_device` data stream could carry information indicating the removal of a referenced device from the network. In the latter case, the `device_status` message would be transmitted to the Broker with a value of false, indicating to the Broker operator the removal of the device from the network. The Broker operator would generate a corresponding message to the Name Server, which would update its table of logical to physical resource mappings accordingly. A value of true on the `device_status` data stream would indicate to the Broker operator the addition of a device to the network, in this case causing the Broker operator to generate a message to the Name Server for the appropriate modifications to its table of logical to physical resource mappings.

Figure 11. Device Server Operator

be transmitted to the Broker with a value of false, indicating to the Broker operator the removal of the device from the network. The Broker operator would generate a corresponding message to the Name Server, which would update its table of logical to physical resource mappings accordingly. A value of true on the `device_status` data stream would indicate to the Broker operator the addition of a device to the network, in this case causing the Broker operator to generate a message to the Name Server for the appropriate modifications to its table of logical to physical resource mappings.
9. **Map Server**

The Map Server, presented in figure 12, also is functionally decomposed into a single atomic operator, locate_map. This operator is triggered upon receipt of the incoming request_map data stream, the contents of which reference a logical map or image requested by the ADSD client. An implementation of the locate_map atomic operator would map the referenced map or image to an actual map or image file and then initiate a file transfer to deliver that file to the ADSD client for display.

![Figure 12. Map Server Operator](image)

10. **Message Server**

The Message Server, presented in figure 13, is functionally decomposed into four atomic operators: three are associated, respectively, with the archiving of incoming messages, requests for messages, and the matching of the two; the fourth atomic operator, template_database, responds to user-generated requests for message templates by searching an internally managed database of templates, then responding on the outgoing data stream, respond_template, with the requested message template.
Figure 13. Message Server Operator
The msg_update_filter atomic operator provides the functionality for the archiving of message filter requests, in the same manner as the alerts_update_filter atomic operator in the Alert Server. Message requests are delivered on the incoming data stream request_msgs, triggering a corresponding state stream msg_filter, which updates the operator itself and notifies the msg_handle_query atomic operator of the new filter request via the msg_filter data stream. The archive_msgs atomic operator provides identical functionality to that described in the update_msgs atomic operator of the Alert Server. It receives incoming messages, generates a self-modifying state stream, msg_message_db, and puts a copy of the new message on the msg_message_db data stream delivered to the msg_handle_query atomic operator. This operator encapsulates the functionality found in the resolve_alerts atomic operator of the Alert Server, which is to match incoming requests for messages against the actual incoming messages themselves. Matches result in an outgoing data stream, respond_msgs, which delivers the matched message to the requesting client via the Broker.

11. Name Server

The Name Server, presented in figure 14, plays a critical role in the distributed network because it represents a single point of reference, mapping logical names to physical locations, facilitating the addition, removal, and location modification of all distributed network system resources. In the SAAWC prototype it is functionally decomposed into a single atomic operator, however, disguising the underlying complexity which would exist in any full implementation of a Name Server, such as the Domain Name
System (DNS), or an X.500 compliant directory service.

The resolve_resource_loc atomic operator is triggered by incoming additions, deletions, and modifications to the locations of system resources which are carried on the incoming data stream mod_obj_loc. State stream mod_obj_loc is generated as a response, and it accomplishes the updating of the internally managed resource database. The incoming data stream, req_resource, represents the client-generated request for a particular resource. The corresponding generated response is the data stream resource_phys_loc which consists of a message indicating the physical location and proper name of the requested resource. This would then be used by the Broker to initiate a method invocation, data type retrieval, or file transfer on behalf of the client in order to
effect the desired results and complete the requested transaction initiated by the corresponding client. An example of such a transaction might be the request for retrieval of a particular map to be displayed by the ADSD client.

12. Print Server

The Print Server, presented in figure 15, is functionally decomposed into three atomic operators. There are potentially two kinds of print requests: those involving files of some type, whether opened for reading or writing or even as temporary files serving as "buffers" or "pipes", and those involving a screen capture or "frame" in a video display. The check_for_file atomic operator receives the incoming print_request data stream and determines which of the two print request types is being referenced. If the print_request references an existing file, then a corresponding request_file data stream, bearing as content the logical name of the requested file, is generated and forwarded to the Broker for name resolution and eventual file location and transfer. The requested file is received on the incoming receive_file data stream, and a corresponding file data stream is then generated, carrying the referenced file to the spool_file atomic operator for subsequent delivery to the appropriate printing device. The spool_file atomic operator generates a corresponding outgoing data stream, job_spooled_msg, which can contain information pertaining to the print job such as the servicing printer, position of the job in the print queue, number of pages in the print job, and so on. In the event that a print request is determined to be referencing a screen grab, the pertinent screen parameters are extracted from the print request message by the check_for_file atomic operator and
Figure 15. Print Server Operator
forwarded on the outgoing screen_param data stream. The build_file atomic operator opens a file for writing and proceeds to create a printable file depicting the referenced display. The file name is forwarded on the outgoing file_name data stream to the spool_file atomic operator where the print request is finally resolved and the print job initiated.

13. Security Server

The Security Server, presented in figure 16, is functionally decomposed into three independent atomic operators: validate_certificate, validate_key, and validate_signature. Implementations of each would validate requests for their respective security mechanisms, search an internally managed database for the appropriate component, and respond on an outgoing data stream with the requested security measure. The respective outgoing data streams for each of the atomic operators are also represented as state streams, which effect modifications to the internally managed databases identifying the issuance of a security measure, the requesting originator, the requested recipient, and other such information as might uniquely describe the particular communication session. Implementations of a Security Server might trigger the issuance of related security measures from the received request of any one component, might contain the functionality to create new security components and discard dated or compromised components, and might also incorporate the functionality to generate alerts to the system administrator in response to unusual requests for security mechanisms.
Figure 16. Security Server Operator
14. Time Server

The Time Server, presented in figure 17, consists of a single atomic operator, update_time. An implementation of this atomic operator would consist of calls to a system time function and the conversion of that returned result into a timestamp which could then be forwarded to the Broker on the outgoing data stream new_time. Though functionally simple, the Time Server provides a critical service by making possible the chronological ordering of all system messages, enabling the Broker to enforce the validation and prioritization rules it has inherited.

15. Track Server

The Track Server, presented in figure 18, consists of three atomic operators functionally aligned with the atomic operators described in the decompositions of the Alert Server and the Message Server. The update_filter atomic operator responds to new track requests contained within the incoming req_tracks data stream by generating a self-modifying state stream, filter, which updates the internally managed database of requested tracks. The receipt of a req_tracks data stream also triggers the release of a
Figure 18. Track Server Operator
message on the outgoing data stream filter, which carries the newly requested track filter to the handle_query operator for matching against existing track records. The update_tracks atomic operator receives new tracks on the incoming feed_tracks data stream, responding with the state stream track_db which modifies the internally managed database of tracks, and with the outgoing data stream track_db, which carries a message to the handle_query atomic operator indicating the existence of the new track. The handle_query atomic operator responds to filter data streams and to track_db data streams by attempting to match requests for tracks with notifications of existing tracks, responding with successful matches on the outgoing data stream respond_tracks. The respond_tracks data stream bears the announcement of a matched track and delivers the message, via the Broker, to the client originating the track request.

E. CAPS PROTOTYPE SYSTEM DESCRIPTION LANGUAGE EDITING

Upon conclusion of the drawing phase of the CAPS prototyping process, the next step was to edit the PSDL code which had been generated by the finished diagrams. All streams designated as state streams in the diagrams, whether to change the state of associated operators or to break existing cycles in the prototype, needed to be identified and declared syntactically within the PSDL code and properly initialized to reflect beginning states. Their declarations as data streams, the default representation for drawn streams, needed to be deleted as well, to prevent duplicate declarations in the PSDL code.

All user-defined data types, contemplated during the drawing phase but first declared in the PSDL editing phase, needed to be syntactically declared and specified as
well. In addition, an operator needed to be specified in the user-defined data type specification to provide default initialization of newly instantiated user-defined objects. All user-defined data types were specified to contain an operator EMPTY, which simply returns a reference to a default initialization, implemented in an Ada source code file, of that particular user-defined data type. For the purposes of this prototype, decisions were made to identify a large number of user-defined data types, and to give them names meant to clarify the intended purpose of those data types. The user-defined data types identified and implemented are: ADMINISTER, ALERT, BITS, CERTIFICATE, DB_RECORD, DEVICE, KEY, MAP, MESSAGE, PARAM, PATH, SIGNATURE, TIMESTAMP, and TRACKS. Figure 19 presents the hierarchy of user-defined data types.

The bottom level contains primitive types and component records which are themselves fields in the composite records of the higher levels. The highest level contains the user-defined data type BITS, which is designed to be the composite type containing system messages, and which is formatted to comply with a specific network datalink protocol. Recognizing the principle of specifying Abstract Data Type (ADT) definitions, but omitting the ADT implementations as unnecessary for the purposes of the prototype, most user-defined data types were envisioned to be simple record types consisting of a primitive data type string component which referenced, perhaps, a configuration file which contained the necessary attribute and method information for the instantiation of the identified data type. For example, a KEY user-defined data type would consist simply of a name field referencing a particular file containing the information necessary to create a
Figure 19. User-defined Data Type Hierarchy

USER-DEFINED DATA TYPE HIERARCHY

MESSAGE

BITS

CERTIFICATE

DEVICE

KEY

MAP

PARAM

PATH

SIGNATURE

composite

atomic

file_name (String)

TIMESTAMP

DB_RECORD

TRACKS

ALERT

ADMINISTER
KEY object, which would then be returned to the KEY request originator for use within the system.

Finally, all operator and data type specifications which were not defined as being implemented by decomposed diagrams or primitive types needed to have their source of implementation identified. This was a simple matter of specifying, in the PSDL code, Ada source code implementations for all atomic operators and user-defined data types.

F. TRANSLATING AND SCHEDULING

Timing constraints and control constraints were not used for the prototype but represent future work for improving the quality of the simulation. As a consequence, the translation process, which produces a compilable Ada source code file to drive the prototype's execution, and the scheduling process, which determines a solution to the problem of scheduling the firing of prototype operators given the set constraints, were enacted and succeeded with no errors.

Future implementation of timing and control constraints would be driven by the compilation and analysis of real world data describing the amount and type of data flow in an actual SAAWC network. For example, data could be collected which documents the number of system-generated tracks or the number of record messages queried and received by a SAAWC operator over a given period of time. Such data could be collected through the monitoring of SAAWC operations during an actual exercise, and could be augmented by the analysis of traffic flow through the network. This latter source of data
is monitored by the communications personnel manning the Communications Systems
Control facility.

G. IMPLEMENTATION

Every user-defined data type and atomic operator identified in the PSDL code
required implementation in a separate Ada source code file. The majority of user-defined
data types were implemented as Ada records, containing a string component which
represents a named reference to a configuration or encoded data file, and a Boolean
component indicating whether the nature of the message containing the data type is a
request or a response. The DB_RECORD and TRACKS data types were defined as Ada
records containing integer, string, and TIMESTAMP components which provide
elaborating information on the physical entities they abstract. The ALERT data type was
defined as an Ada record which references DB_RECORDS and TRACKS and provides
amplifying information in the form of integer and string components which provide
location and time context for the referenced DB_RECORDS and TRACKS. Finally, the
MESSAGE data type was implemented as an Ada record referencing all other non-BIT
user-defined data types. Its role in the prototype is analogous to that of a
network-specific protocol packet, representing the atomic unit upon which all atomic
operators designed to function as interfaces within the network can properly perform
receipt, processing, and repackaging operations.

All atomic operators were implemented with the simple functionality of invoking
the system stream output operator to print operator-specific messages to the prototype
console text window. Future implementations of the atomic operators would either use off the shelf components to provide the requisite functionality or skeleton code to simulate the expected behavior of the operator. For example, the atomic operators designated to encapsulate the functionality of the DBMS module executing record deletions could be implemented with a commercially available DBMS product or by a simple linked list implementation which effects a runtime-only simulation of database management behavior.

H. SUMMARY

The analysis, design, and implementation of the SAAWC prototype in CAPS involved a number of computing tools and document resources. Satisfaction with the correct, desired, logical decomposition of the model was the result of numerous drawings and protracted analysis with drawing and spreadsheet tools. Familiarization with CAPS itself came with the review of available tutorials and user manuals, and with experimentation on small-scale prototypes. Improvements to the prototype design often came in the wake of articulations of the design philosophy in writing, which occurred during the concurrent editing of this document. Finally, refinement of the prototype design resulted from numerous critical comments from peers and advisors.
V. CONCLUSION

A. SAAWC PROTOTYPE SIGNIFICANCE

The need exists to identify redundancy in the development of automated information systems for employment in the conduct of command, control, communication, computing, and intelligence operations both in peace and in time of conflict. The prerequisite for this identification is a thorough examination and analysis of the data types and processes, both common support and task-specific, which exist in a particular operational domain, such as the Sector Anti Air Warfare Center. The SAAWC is a pertinent candidate and an important model for this analysis, as it incorporates much of the functionality exhibited by C4I workstations: consumption of real-time and non-real-time data, dynamic display of operational events in the context of the time and space which they occupy, redundant means of communication incorporating text, graphic, audio, and video representations, powerful rules-based analysis through the identification and presentation of event abstractions and elaborating documentation. Comprehensive and accurate identification of the data types and processes which will enable the functionality identified as required by system operators is a critical first step in the design and implementation of the strictly defined software modules essential to the often touted vision of "plug and play." Comprehensive analysis of these data types and processes will permit the development of well-defined interfaces, which in turn will permit the independent and concurrent development of the common support and task-specific implementations which will satisfy the functional needs of the operator.
B. CAPS PROTOTYPING

Prototyping is a quick, low-risk, cost-effective solution to the problem of developing automated information systems which augment and, increasingly, enable the conduct of C4I operations. The NPS Computer Aided Prototyping System prototyping of the SAAWC facilitated a deliberate, logical analysis of the atomic functionality required to provide those services identified in the SAAWC reference manual as well as those which are in compliance with the DII COE, GCCS, and JMCIS architectural guidelines. CAPS provides an integrated set of tools which permit the specification, design, and implementation of a prototype to occur within a single integrated development environment. CAPS also provides the functionality to propagate changes made to the prototype design in the graphical editor both to the PSDL specification and to the source code file which drives the executing prototype. Together with the integrated schedule writing module, these CAPS capabilities increase the likelihood that a developer attempting to model a distributed, networked system will be able to create a simulation which more closely fulfills the needs of the operational community.

C. FUTURE WORK

The work completed on the CAPS SAAWC prototype leaves open the possibility of future work on several levels. The basic upper level SAAWC decomposition could be used with another SAAWC functional area, for example the Air Defense Mission Display, replacing the Air Defense Situation Display operator. The upper level decomposition could be used with some completely unrelated functional operator, such as the Intelligence
Analysis Display in the Tactical Air Command Center, replacing the ADSD operator. Additional composite operators might be identified to provide common services either partially provided for or omitted in the current upper level decomposition. Such operators might include a directory service, a resource scheduler service, or a network management service. Composite operators currently identified in the upper level diagram might be decomposed differently to identify additional functionality or to provide a greater level of detail in the currently identified functionality. For example, the update_filter operator in the Track Server composite operator might be decomposed to convey functionality which determines whether incoming filter requests are redundant or improperly initialized. Timing constraints and control constraints could be used within designated operators to construct a prototype which more accurately behaves like the real-world system it is simulating. Such behavior could be examined to identify the nodes and datalinks which present potential system bottlenecks, or which require exceptional processing and secondary storage resources. Finally, some or all user-defined data types and atomic operators could be implemented with commercially available components or by improving the existing Ada source code to more closely resemble the behavior of the simulated system.
APPENDIX A

SAAWC PROTOTYPE SOURCE CODE
<table>
<thead>
<tr>
<th>Column 1</th>
<th>Column 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Column 3</td>
<td>Column 4</td>
</tr>
<tr>
<td>Column 5</td>
<td>Column 6</td>
</tr>
<tr>
<td>Column 7</td>
<td>Column 8</td>
</tr>
<tr>
<td>Column 9</td>
<td>Column 10</td>
</tr>
</tbody>
</table>

**Table Example**

- **Column 1**: Various entries related to vertex management and record types.
- **Column 2**: Additional information placeholder.
- **Column 3**: Placeholder with potential data entries.
- **Column 4**: Placeholder with potential data entries.
- **Column 5**: Placeholder with potential data entries.
- **Column 6**: Placeholder with potential data entries.
- **Column 7**: Placeholder with potential data entries.
- **Column 8**: Placeholder with potential data entries.
- **Column 9**: Placeholder with potential data entries.
- **Column 10**: Placeholder with potential data entries.
INITIAL

START

INITIAL

STATE

INITIAL

STATE

INITIAL

STATE

INITIAL

STATE

INITIAL

STATE

INITIAL

STATE

INITIAL

STATE

INITIAL

STATE

INITIAL

STATE

INITIAL

STATE

INITIAL

STATE

INITIAL

STATE

INITIAL

STATE

INITIAL

STATE

INITIAL

STATE

INITIAL

STATE

INITIAL

STATE

INITIAL

STATE

INITIAL

STATE

INITIAL

STATE

INITIAL

STATE

INITIAL

STATE

INITIAL

STATE

INITIAL

STATE

INITIAL

STATE

INITIAL

STATE

INITIAL

STATE

INITIAL

STATE

INITIAL

STATE

INITIAL

STATE

INITIAL

STATE

INITIAL

STATE

INITIAL

STATE

INITIAL

STATE

INITIAL

STATE

INITIAL

STATE

INITIAL

STATE

INITIAL

STATE

INITIAL

STATE

INITIAL

STATE

INITIAL

STATE

INITIAL

STATE

INITIAL

STATE

INITIAL

STATE

INITIAL

STATE

INITIAL

STATE

INITIAL

STATE

INITIAL

STATE

INITIAL

STATE

INITIAL

STATE

INITIAL

STATE

INITIAL

STATE

INITIAL

STATE

INITIAL

STATE

INITIAL

STATE

INITIAL

STATE

INITIAL

STATE

INITIAL

STATE

INITIAL

STATE

INITIAL

STATE

INITIAL

STATE

INITIAL

STATE

INITIAL

STATE

INITIAL

STATE

INITIAL

STATE

INITIAL

STATE

INITIAL

STATE

INITIAL

STATE

INITIAL

STATE

INITIAL

STATE

INITIAL

STATE

INITIAL

STATE

INITIAL

STATE

INITIAL

STATE

INITIAL

STATE

INITIAL

STATE

INITIAL

STATE

INITIAL

STATE

INITIAL

STATE

INITIAL

STATE

INITIAL

STATE

INITIAL

STATE

INITIAL

STATE

INITIAL

STATE

INITIAL

STATE

INITIAL

STATE

INITIAL

STATE

INITIAL

STATE

INITIAL

STATE

INITIAL

STATE

INITIAL

STATE

INITIAL

STATE

INITIAL

STATE

INITIAL

STATE

INITIAL

STATE

INITIAL

STATE

INITIAL

STATE

INITIAL

STATE

INITIAL

STATE

INITIAL

STATE

INITIAL

STATE

INITIAL

STATE

INITIAL

STATE

INITIAL

STATE

INITIAL

STATE

INITIAL

STATE

INITIAL

STATE

INITIAL

STATE

INITIAL

STATE

INITIAL

STATE

INITIAL

STATE

INITIAL

STATE

INITIAL

STATE

INITIAL

STATE

INITIAL

STATE

INITIAL

STATE

INITIAL

STATE

INITIAL

STATE

INITIAL

STATE

INITIAL

STATE

INITIAL

STATE

INITIAL

STATE

INITIAL

STATE

INITIAL

STATE

INITIAL

STATE

INITIAL

STATE

INITIAL

STATE

INITIAL

STATE

INITIAL

STATE

INITIAL

STATE

INITIAL

STATE

INITIAL

STATE

INITIAL

STATE

INITIAL

STATE

INITIAL

STATE

INITIAL

STATE

INITIAL

STATE

INITIAL

STATE

INITIAL

STATE

INITIAL

STATE

INITIAL

STATE

INITIAL

STATE

INITIAL

STATE

INITIAL

STATE

INITIAL

STATE

INITIAL

STATE

INITIAL

STATE

INITIAL

STATE

INITIAL

STATE

INITIAL

STATE

INITIAL

STATE

INITIAL

STATE

INITIAL

STATE

INITIAL

STATE

INITIAL

STATE

INITIAL

STATE

INITIAL

STATE

INITIAL

STATE

INITIAL

STATE

INITIAL

STATE

INITIAL

STATE

INITIAL

STATE

INITIAL

STATE

INITIAL

STATE

INITIAL

STATE

INITIAL

STATE

INITIAL

STATE

INITIAL

STATE

INITIAL

STATE

INITIAL

STATE

INITIAL

STATE

INITIAL

STATE

INITIAL

STATE

INITIAL

STATE

INITIAL

STATE

INITIAL

STATE

INITIAL

STATE

INITIAL

STATE

INITIAL

STATE

INITIAL

STATE

INITIAL

STATE

INITIAL

STATE

INITIAL

STATE

INITIAL

STATE

INITIAL

STATE

INITIAL

STATE

INITIAL

STATE

INITIAL

STATE
In the context of the task, the development of a technology-based approach was discussed. This approach is designed to improve the current methods by integrating advanced features and functionalities. The key aspects include

- Integration of sensor data
- Optimization of signal processing
- Enhanced communication protocols

These features are crucial for creating a more efficient and reliable system. Furthermore, the approach involves the use of machine learning algorithms to analyze data and make predictions. This allows for proactive decision-making, reducing the need for manual intervention.

In conclusion, the technology-based approach offers significant advancements in the field. It not only improves current capabilities but also paves the way for future innovations. Therefore, it is essential to continue researching and developing such technologies to stay ahead in the competition.
null
EXCEPTION HANDLER 1

EXCEPTION HANDLER 2

EXCEPTION HANDLER 3

EXCEPTION HANDLER 4

EXCEPTION HANDLER 5

EXCEPTION HANDLER 6

EXCEPTION HANDLER 7

EXCEPTION HANDLER 8

EXCEPTION HANDLER 9

EXCEPTION HANDLER 10

EXCEPTION HANDLER 11

EXCEPTION HANDLER 12

EXCEPTION HANDLER 13

EXCEPTION HANDLER 14

EXCEPTION HANDLER 15

EXCEPTION HANDLER 16

EXCEPTION HANDLER 17

EXCEPTION HANDLER 18

EXCEPTION HANDLER 19

EXCEPTION HANDLER 20

EXCEPTION HANDLER 21

EXCEPTION HANDLER 22

EXCEPTION HANDLER 23

EXCEPTION HANDLER 24

EXCEPTION HANDLER 25

EXCEPTION HANDLER 26

EXCEPTION HANDLER 27

EXCEPTION HANDLER 28

EXCEPTION HANDLER 29

EXCEPTION HANDLER 30

EXCEPTION HANDLER 31

EXCEPTION HANDLER 32

EXCEPTION HANDLER 33

EXCEPTION HANDLER 34

EXCEPTION HANDLER 35

EXCEPTION HANDLER 36

EXCEPTION HANDLER 37

EXCEPTION HANDLER 38

EXCEPTION HANDLER 39

EXCEPTION HANDLER 40

EXCEPTION HANDLER 41

EXCEPTION HANDLER 42

EXCEPTION HANDLER 43

EXCEPTION HANDLER 44

EXCEPTION HANDLER 45

EXCEPTION HANDLER 46

EXCEPTION HANDLER 47

EXCEPTION HANDLER 48

EXCEPTION HANDLER 49

EXCEPTION HANDLER 50

EXCEPTION HANDLER 51

EXCEPTION HANDLER 52

EXCEPTION HANDLER 53

EXCEPTION HANDLER 54

EXCEPTION HANDLER 55

EXCEPTION HANDLER 56

EXCEPTION HANDLER 57

EXCEPTION HANDLER 58

EXCEPTION HANDLER 59

EXCEPTION HANDLER 60

EXCEPTION HANDLER 61

EXCEPTION HANDLER 62

EXCEPTION HANDLER 63

EXCEPTION HANDLER 64

EXCEPTION HANDLER 65

EXCEPTION HANDLER 66

EXCEPTION HANDLER 67

EXCEPTION HANDLER 68

EXCEPTION HANDLER 69

EXCEPTION HANDLER 70

EXCEPTION HANDLER 71

EXCEPTION HANDLER 72

EXCEPTION HANDLER 73

EXCEPTION HANDLER 74

EXCEPTION HANDLER 75

EXCEPTION HANDLER 76

EXCEPTION HANDLER 77

EXCEPTION HANDLER 78

EXCEPTION HANDLER 79

EXCEPTION HANDLER 80

EXCEPTION HANDLER 81

EXCEPTION HANDLER 82

EXCEPTION HANDLER 83

EXCEPTION HANDLER 84

EXCEPTION HANDLER 85

EXCEPTION HANDLER 86

EXCEPTION HANDLER 87

EXCEPTION HANDLER 88

EXCEPTION HANDLER 89

EXCEPTION HANDLER 90

EXCEPTION HANDLER 91

EXCEPTION HANDLER 92

EXCEPTION HANDLER 93

EXCEPTION HANDLER 94

EXCEPTION HANDLER 95

EXCEPTION HANDLER 96

EXCEPTION HANDLER 97

EXCEPTION HANDLER 98

EXCEPTION HANDLER 99

EXCEPTION HANDLER 100

EXCEPTION HANDLER 101
and ADD FILE PKG:
and ADD FILE:
begin
  PROCEDURE ADD FILE (REG, ADD FILE: IN DB-RECORD; FILE-NAME; OUT FILE-DESC) IS
  PROCEDURE ADD FILE (REG, ADD FILE: IN DB-RECORD; FILE-NAME; OUT FILE-DESC) IS
  PROCEDURE ADD FILE (REG, ADD FILE: IN DB-RECORD; FILE-NAME; OUT FILE-DESC) IS
  PROCEDURE ADD FILE (REG, ADD FILE: IN DB-RECORD; FILE-NAME; OUT FILE-DESC) IS
  with DB-RECORD, FILE-DESC:
  with DB-RECORD, FILE-DESC:
  with DB-RECORD, FILE-DESC:
  with DB-RECORD, FILE-DESC:
--Purpose: This package adds DB-RECORDs to database
--Project: {other - A CAPP Progress Of The Same
--Author: {other HW
--Engineer: Simon Add 11/05/99

procedure ADTAccord (REG, ACCRec, in DBAccord, ACCRec, out DBAccord) is
package body ADTAccord is
package ADTAccord is
with DBAccord, Excel, DBAccord, Excel;
with Excel;--Access Data Base Reor regulates database
--purpose: to create procedures of the sample
--Date: 12 June 1997
--Author: Carme Morey, C.L.
end ADMINISTER-
end ENTR

dump 'administer name= 'FAKERS"
begin

dump 'ADMINISTER

function ENTR return ADMINISTER
package body ADMINISTER

end ADMINISTER

function ENTR return ADMINISTER
begin

package ADMINISTER

package ADMINISTER

end ADMINISTER

use this name

with this

end for

---

---

---
and bits~PROG!
and ENTRY!
return dummy!
begin
dummy : BITS!
function carry return BITS is
package body BITS~prog is
end BITS~prog;
function carry return BITS is
end record;
package message is
rec message := MESSAGE type
end package message is
with MESSAGE.prog is
with MESSAGE.prog is
with MESSAGE.prog is
...
173

begin

VALID-RESPONSE (out MESSAGE)

if MESSAGE-READY (in MESSAGE) then

procedure BUSINESS-RULES-PROCESS (CLIENT; MESSAGE)

end BUSINESS-RULES-PROCESS

end MESSAGE-

VALID-RESPONSE (out MESSAGE)

end BUSINESS-RULES-PROCESS

end BUSINESS-RULES-PROCESS

end MESSAGE-READY (in MESSAGE)

end MESSAGE-READY (in MESSAGE)

end MESSAGE-READY (in MESSAGE)

end MESSAGE-READY (in MESSAGE)

end MESSAGE-READY (in MESSAGE)

end MESSAGE-READY (in MESSAGE)

end MESSAGE-READY (in MESSAGE)

end MESSAGE-READY (in MESSAGE)

end MESSAGE-READY (in MESSAGE)

end MESSAGE-READY (in MESSAGE)

end MESSAGE-READY (in MESSAGE)

end MESSAGE-READY (in MESSAGE)

end MESSAGE-READY (in MESSAGE)

end MESSAGE-READY (in MESSAGE)

end MESSAGE-READY (in MESSAGE)

end MESSAGE-READY (in MESSAGE)

end MESSAGE-READY (in MESSAGE)

end MESSAGE-READY (in MESSAGE)

end MESSAGE-READY (in MESSAGE)

end MESSAGE-READY (in MESSAGE)

end MESSAGE-READY (in MESSAGE)

end MESSAGE-READY (in MESSAGE)

end MESSAGE-READY (in MESSAGE)

end MESSAGE-READY (in MESSAGE)

end MESSAGE-READY (in MESSAGE)

end MESSAGE-READY (in MESSAGE)

end MESSAGE-READY (in MESSAGE)

end MESSAGE-READY (in MESSAGE)

end MESSAGE-READY (in MESSAGE)

end MESSAGE-READY (in MESSAGE)

end MESSAGE-READY (in MESSAGE)

end MESSAGE-READY (in MESSAGE)

end MESSAGE-READY (in MESSAGE)

end MESSAGE-READY (in MESSAGE)

end MESSAGE-READY (in MESSAGE)

end MESSAGE-READY (in MESSAGE)

end MESSAGE-READY (in MESSAGE)

end MESSAGE-READY (in MESSAGE)

end MESSAGE-READY (in MESSAGE)

end MESSAGE-READY (in MESSAGE)

end MESSAGE-READY (in MESSAGE)

end MESSAGE-READY (in MESSAGE)

end MESSAGE-READY (in MESSAGE)

end MESSAGE-READY (in MESSAGE)

end MESSAGE-READY (in MESSAGE)

end MESSAGE-READY (in MESSAGE)

end MESSAGE-READY (in MESSAGE)

end MESSAGE-READY (in MESSAGE)

end MESSAGE-READY (in MESSAGE)

end MESSAGE-READY (in MESSAGE)

end MESSAGE-READY (in MESSAGE)

end MESSAGE-READY (in MESSAGE)

end MESSAGE-READY (in MESSAGE)

end MESSAGE-READY (in MESSAGE)

end MESSAGE-READY (in MESSAGE)

end MESSAGE-READY (in MESSAGE)

end MESSAGE-READY (in MESSAGE)

end MESSAGE-READY (in MESSAGE)

end MESSAGE-READY (in MESSAGE)

end MESSAGE-READY (in MESSAGE)

end MESSAGE-READY (in MESSAGE)

end MESSAGE-READY (in MESSAGE)

end MESSAGE-READY (in MESSAGE)

end MESSAGE-READY (in MESSAGE)

end MESSAGE-READY (in MESSAGE)

end MESSAGE-READY (in MESSAGE)

end MESSAGE-READY (in MESSAGE)

end MESSAGE-READY (in MESSAGE)

end MESSAGE-READY (in MESSAGE)

end MESSAGE-READY (in MESSAGE)

end MESSAGE-READY (in MESSAGE)

end MESSAGE-READY (in MESSAGE)

end MESSAGE-READY (in MESSAGE)

end MESSAGE-READY (in MESSAGE)

end MESSAGE-READY (in MESSAGE)

end MESSAGE-READY (in MESSAGE)

end MESSAGE-READY (in MESSAGE)

end MESSAGE-READY (in MESSAGE)

end MESSAGE-READY (in MESSAGE)

end MESSAGE-READY (in MESSAGE)

end MESSAGE-READY (in MESSAGE)

end MESSAGE-READY (in MESSAGE)

end MESSAGE-READY (in MESSAGE)

end MESSAGE-READY (in MESSAGE)

end MESSAGE-READY (in MESSAGE)

end MESSAGE-READY (in MESSAGE)

end MESSAGE-READY (in MESSAGE)

end MESSAGE-READY (in MESSAGE)

end MESSAGE-READY (in MESSAGE)

end MESSAGE-READY (in MESSAGE)

end MESSAGE-READY (in MESSAGE)

end MESSAGE-READY (in MESSAGE)

end MESSAGE-READY (in MESSAGE)

end MESSAGE-READY (in MESSAGE)

end MESSAGE-READY (in MESSAGE)

end MESSAGE-READY (in MESSAGE)

end MESSAGE-READY (in MESSAGE)

end MESSAGE-READY (in MESSAGE)

end MESSAGE-READY (in MESSAGE)

end MESSAGE-READY (in MESSAGE)

end MESSAGE-READY (in MESSAGE)

end MESSAGE-READY (in MESSAGE)

end MESSAGE-READY (in MESSAGE)

end MESSAGE-READY (in MESSAGE)

end MESSAGE-READY (in MESSAGE)

end MESSAGE-READY (in MESSAGE)

end MESSAGE-READY (in MESSAGE)

end MESSAGE-READY (in MESSAGE)

end MESSAGE-READY (in MESSAGE)

end MESSAGE-READY (in MESSAGE)

end MESSAGE-READY (in MESSAGE)

end MESSAGE-READY (in MESSAGE)

end MESSAGE-READY (in MESSAGE)

end MESSAGE-READY (in MESSAGE)

end MESSAGE-READY (in MESSAGE)

end MESSAGE-READY (in MESSAGE)

end MESSAGE-READY (in MESSAGE)

end MESSAGE-READY (in MESSAGE)

end MESSAGE-READY (in MESSAGE)

end MESSAGE-READY (in MESSAGE)

end MESSAGE-READY (in MESSAGE)

end MESSAGE-READY (in MESSAGE)

end MESSAGE-READY (in MESSAGE)

end MESSAGE-READY (in MESSAGE)

end MESSAGE-READY (in MESSAGE)

end MESSAGE-READY (in MESSAGE)

end MESSAGE-READY (in MESSAGE)

end MESSAGE-READY (in MESSAGE)

end MESSAGE-READY (in MESSAGE)

end MESSAGE-READY (in MESSAGE)

end MESSAGE-READY (in MESSAGE)

end MESSAGE-READY (in MESSAGE)

end MESSAGE-READY (in MESSAGE)

end MESSAGE-READY (in MESSAGE)

end MESSAGE-READY (in MESSAGE)

end MESSAGE-READY (in MESSAGE)

end MESSAGE-READY (in MESSAGE)

end MESSAGE-READY (in MESSAGE)

end MESSAGE-READY (in MESSAGE)

end MESSAGE-READY (in MESSAGE)

end MESSAGE-READY (in MESSAGE)

end MESSAGE-READY (in MESSAGE)

end MESSAGE-READY (in MESSAGE)

end MESSAGE-READY (in MESSAGE)

end MESSAGE-READY (in MESSAGE)

end MESSAGE-READY (in MESSAGE)

end MESSAGE-READY (in MESSAGE)

end MESSAGE-READY (in MESSAGE)

end MESSAGE-READY (in MESSAGE)

end MESSAGE-READY (in MESSAGE)

end MESSAGE-READY (in MESSAGE)

end MESSAGE-READY (in MESSAGE)

end MESSAGE-READY (in MESSAGE)

end MESSAGE-READY (in MESSAGE)

end MESSAGE-READY (in MESSAGE)

end MESSAGE-READY (in MESSAGE)

end MESSAGE-READY (in MESSAGE)

end MESSAGE-READY (in MESSAGE)

end MESSAGE-READY (in MESSAGE)

end MESSAGE-READY (in MESSAGE)

end MESSAGE-READY (in MESSAGE)

end MESSAGE-READY (in MESSAGE)

end MESSAGE-READY (in MESSAGE)

end MESSAGE-READY (in MESSAGE)

end MESSAGE-READY (in MESSAGE)

end MESSAGE-READY (in MESSAGE)

end MESSAGE-READY (in MESSAGE)

end MESSAGE-READY (in MESSAGE)

end MESSAGE-READY (in MESSAGE)

end MESSAGE-READY (in MESSAGE)

end MESSAGE-READY (in MESSAGE)

end MESSAGE-READY (in MESSAGE)

end MESSAGE-READY (in MESSAGE)

end MESSAGE-READY (in MESSAGE)

end MESSAGE-READY (in MESSAGE)
begin

SCREEN: Align our Blake is
FILE: or PART: Replace FILE: our MESSAGE


package body CHECK: FOR: FILE: PRG

end CHECK: FOR: FILE: PRG

SCREEN: Align our Blake is
FILE: or PART: Replace FILE: our MESSAGE


package CHECK: FOR: FILE: PRG

with PROCESS: PRINT: FILE: PRG

with MESSAGE: PRINT: MESSAGE: PRG

with text: "Logon User: 101"

-- or need: to be built
-- purpose: this package recieves a print request and determines if the exte
-- package: files 4 and process: of the same
-- date: 3 July 1997
-- author: cask hill kowni
-- formation: smaw: check: file: 1
and client-thread-manager-free

Pre-iteration => Inside operator client-thread-manager-free:

package body client-thread-manager-free is

and client-thread-manager-free end

call post-iteration vs client-thread-manager-free end

package body client-thread-manager-free is

and client-thread-manager-free end

end client-thread-manager-free

-- Package: This package tracks and handles all messages from clients
-- Procedure: A CALL procedure on the given
-- Author: Carl Harrington
-- ITU: 91-000-01
PROCEDURE CORRECT-UPDATE-TRACKS

PROCESS: correct-update-tracks

package body correct-update-tracks

begin

if true then

procedure correct-update-tracks : in tracks; correct-tracks : out tracks

package body correct-update-tracks

begin

procedure correct-update-tracks : in tracks; correct-tracks : out tracks

package body correct-update-tracks

with tracks, package update-tracks

begin

with tracks, package update-tracks

end update-tracks

end correct-update-tracks

end package body correct-update-tracks

end if

end correct-update-tracks

end PROCEDURE CORRECT-UPDATE-TRACKS
end delcette-package prog

end delcette-package

pur-line (term = "Initiate operation multiple times")

begin

procedure delcette-package (src-record in del-record, & del-record out del-record)

package body delcette-package prog is

procedure delcette-package prog is

with del-record-package del-record & del-record prog;

with del-record-package del-record prog;

package del-record-package (del-record prog)

--purpose: This package deletes del-record tuples from database
--package: del-record
--data structure: del-record
--author: ?? date: ??

package del-record-package (del-record prog)

--purpose: This package deletes del-record tuples from database
--package: del-record
--data structure: del-record
--author: ?? date: ??


end device-
end global;

return dump;

dump .device name is "OTHERS" 

begin

device: DEVICE;

function EMPTY return device is

package body DEVICE

end package;

function EMPTY return device

end function;

device name: title name;

type device is record

package device; 
x is

use title name;

with title name;

use record;

with record;

--This file is where I declare the type device

--Pragma:

--Project: SHARE 26/07/1989
--Hardware: CPE7922
--Monitor: CPE 7922
--Compiler: SHARE-CPE26

--If you are not using the SHARE environment
begin

procedure TPPROCESSOR1(IN: MESSAGE; OUT: TPPROCESSOR1);

package body TPPROCESSOR1 is

procedure TPPROCESSOR1(IN: MESSAGE; OUT: TPPROCESSOR1) is

end TPPROCESSOR1;

end TPPROCESSOR1;

-- Purpose: This package processes outgoing and incoming messages
-- Project: THESIS - A CASE PROJECT OF THE AUTHOR
-- Date: 17 June 1997
-- Author: C. M. Kowall

-- Interface: See the package TPPROCESSOR1
and GET-USER-CMD?

and GET-USER-CMD?

begin

PACKAGE body include 1.

PACKAGE body include 1.

begin

begin

PACKAGE body include 1.

PACKAGE body include 1.

begin

begin

PACKAGE body include 1.

PACKAGE body include 1.

begin

begin

PACKAGE body include 1.

PACKAGE body include 1.

begin

begin

PACKAGE body include 1.

PACKAGE body include 1.

begin

begin

PACKAGE body include 1.

PACKAGE body include 1.

begin

begin

PACKAGE body include 1.

PACKAGE body include 1.

begin

begin

PACKAGE body include 1.

PACKAGE body include 1.

begin

begin

PACKAGE body include 1.

PACKAGE body include 1.

begin

begin

PACKAGE body include 1.

PACKAGE body include 1.

begin

begin

PACKAGE body include 1.

PACKAGE body include 1.

begin

begin

PACKAGE body include 1.

PACKAGE body include 1.

begin

begin

PACKAGE body include 1.

PACKAGE body include 1.

begin

begin

PACKAGE body include 1.

PACKAGE body include 1.

begin

begin

PACKAGE body include 1.

PACKAGE body include 1.

begin

begin

PACKAGE body include 1.

PACKAGE body include 1.

begin

begin

PACKAGE body include 1.

PACKAGE body include 1.

begin

begin

PACKAGE body include 1.

PACKAGE body include 1.

begin

begin

PACKAGE body include 1.

PACKAGE body include 1.

begin

begin

PACKAGE body include 1.

PACKAGE body include 1.

begin

begin

PACKAGE body include 1.

PACKAGE body include 1.

begin

begin

PACKAGE body include 1.

PACKAGE body include 1.

begin

begin

PACKAGE body include 1.

PACKAGE body include 1.

begin

begin

PACKAGE body include 1.

PACKAGE body include 1.

begin

begin

PACKAGE body include 1.

PACKAGE body include 1.

begin

begin

PACKAGE body include 1.

PACKAGE body include 1.

begin

begin

PACKAGE body include 1.

PACKAGE body include 1.

begin

begin

PACKAGE body include 1.

PACKAGE body include 1.

begin

begin

PACKAGE body include 1.

PACKAGE body include 1.

begin

begin

PACKAGE body include 1.

PACKAGE body include 1.

begin

begin

PACKAGE body include 1.

PACKAGE body include 1.

begin

begin

PACKAGE body include 1.

PACKAGE body include 1.

begin

begin

PACKAGE body include 1.

PACKAGE body include 1.

begin

begin

PACKAGE body include 1.

PACKAGE body include 1.

begin

begin

PACKAGE body include 1.

PACKAGE body include 1.

begin

begin

PACKAGE body include 1.

PACKAGE body include 1.

begin

begin

PACKAGE body include 1.

PACKAGE body include 1.

begin

begin

PACKAGE body include 1.

PACKAGE body include 1.

begin

begin

PACKAGE body include 1.

PACKAGE body include 1.

begin

begin

PACKAGE body include 1.

PACKAGE body include 1.

begin

begin

PACKAGE body include 1.

PACKAGE body include 1.

begin

begin

PACKAGE body include 1.

PACKAGE body include 1.

begin

begin

PACKAGE body include 1.

PACKAGE body include 1.

begin

begin

PACKAGE body include 1.

PACKAGE body include 1.

begin

begin

PACKAGE body include 1.

PACKAGE body include 1.

begin

begin

PACKAGE body include 1.

PACKAGE body include 1.

begin

begin

PACKAGE body include 1.

PACKAGE body include 1.

begin

begin

PACKAGE body include 1.

PACKAGE body include 1.

begin

begin

PACKAGE body include 1.

PACKAGE body include 1.

begin

begin

PACKAGE body include 1.

PACKAGE body include 1.

begin

begin

PACKAGE body include 1.

PACKAGE body include 1.

begin

begin

PACKAGE body include 1.

PACKAGE body include 1.

begin

begin

PACKAGE body include 1.
begin

...
and MAP->%f;
and FT-
:RETURN:
RETURN Dummy:
\n dummy::mp:mp renamed [.entries] = entry
begin
dummy := entry:
\n function EMPTY RETURN MAP
package body entry at
end MAP->%
function EMPTY RETURN MAP;
end record;
required : boolean = true;
map:name = entry::name;
\n empty MAP->%f:
package body entry at
empty [type] name:
with entry::name; end
with entry::name; end
with entry::name; end

-This title is where I declare the type MAP-

-Purpose:
function: that is a part of the prototype of the statement

-program: 02 Jun 97
-authors: see last comment
-lemmas: some comments professions
BEGIN

PROCEDURE MOD-FILE (MOD-FILE IN DB-RECORD, ND-RECORD OUT DB-RECORD)

PROCEDURE MOD-FILE (MOD-FILE IN DB-RECORD, ND-RECORD OUT DB-RECORD)

PROCEDURE MOD-FILE (MOD-FILE IN DB-RECORD, ND-RECORD OUT DB-RECORD)

with DB-RECORD, Project DB-RECORD, MOD.

-- Purpose: This package modifies DB-RECORD types in database
-- Project: mom - A CAPS PROCEDURE OF THE DANCE
-- Author: CAPS (CAPS) NAME
-- Version: 1.0 June 1999

end MOD-FILE;

end MOD-FILE.
package MGO_OBJECT_PKG is

procedure KSL_OBJECT_PKG (req, MD, OBJ, IN DB-RECORD, OUT DB-RECORD) is

end MGO_OBJECT_PKG;

end MGO_OBJECT。

procedure MGO_OBJECT_PKG (req, MD, OBJ, IN DB-RECORD, OUT DB-RECORD) is

begin

end MGO_OBJECT_PKG;

end MGO_OBJECT。

package body MGO_OBJECT_PKG is

procedure KSL_OBJECT_PKG (req, MD, OBJ, IN DB-RECORD, OUT DB-RECORD) is

end MGO_OBJECT_PKG;
end mode ←record.end
end mode ←record

begin
procedure mode ←record(start: in char; end: in char; file: out record)
package body mode ←record is

procedure mode ←record(start: in char; end: in char; file: out record)
package body mode ←record is

with mode ←record package body mode ←record
with mode ←record package body mode ←record

--purpose: this package models DB− records types in database
--project: project − a DB−preprocessor of the SANE
--version: 1.2 June 1997
--authors: Ralf Rolfke

begin

procedure msg.UpdateFilter(in Msg : in Message; out Filter : out Message)
begin
package body msg.UpdateFilter
end msg.UpdateFilter;

end msg.UpdateFilter;

end Package.

Purpose: This package contains a message filter procedure for
Project: "t" - A Data Processing of the Scenario
--Created: 1 July 1999
--Author: Capsule Homes
--Filename: smc-update-filter.a
and NETWORK-PACK

end NETWORK-

procedure NETWORK-BITE
call 1 BITE-IN ; OUT BITE
end NETWORK-BITE

procedure NETWORK-PACK is

with BODY-PACK; use BODY-PACK;

end NETWORK-PACK;
and return Path;
and return;

function EMPTY return Path is
begin

return Path;

function EMPTY return Path is
begin

return Path;

record;

parameter : boolean := True;

package Path is record

package Path is record

use (11) package Path;

use (11) package Path;

with (11) package Path:

with (11) package Path:

with (11) package Path:

with (11) package Path:

with (11) package Path:

with (11) package Path:

with (11) package Path:

with (11) package Path:

with (11) package Path:

with (11) package Path:

with (11) package Path:

with (11) package Path:

with (11) package Path:

with (11) package Path:

with (11) package Path:

with (11) package Path:

with (11) package Path:

with (11) package Path:

with (11) package Path:

with (11) package Path:

with (11) package Path:

with (11) package Path:

with (11) package Path:

with (11) package Path:

with (11) package Path:

with (11) package Path:

with (11) package Path:

with (11) package Path:

with (11) package Path:

with (11) package Path:

with (11) package Path:

with (11) package Path:

with (11) package Path:

with (11) package Path:

with (11) package Path:

with (11) package Path:

with (11) package Path:

with (11) package Path:

with (11) package Path:

with (11) package Path:

with (11) package Path:

with (11) package Path:

with (11) package Path:

with (11) package Path:

with (11) package Path:

with (11) package Path:

with (11) package Path:

with (11) package Path:

with (11) package Path:

with (11) package Path:

with (11) package Path:

with (11) package Path:

with (11) package Path:

with (11) package Path:

with (11) package Path:

with (11) package Path:

with (11) package Path:

with (11) package Path:

with (11) package Path:

with (11) package Path:

with (11) package Path:

with (11) package Path:

with (11) package Path:

with (11) package Path:

with (11) package Path:

with (11) package Path:

with (11) package Path:

with (11) package Path:

with (11) package Path:

with (11) package Path:

with (11) package Path:

with (11) package Path:

with (11) package Path:

with (11) package Path:

with (11) package Path:

with (11) package Path:

with (11) package Path:

with (11) package Path:

with (11) package Path:

with (11) package Path:

with (11) package Path:

with (11) package Path:

with (11) package Path:

with (11) package Path:

with (11) package Path:

with (11) package Path:

with (11) package Path:

with (11) package Path:

with (11) package Path:

with (11) package Path:

with (11) package Path:

with (11) package Path:

with (11) package Path:

with (11) package Path:

with (11) package Path:

with (11) package Path:

with (11) package Path:

with (11) package Path:

with (11) package Path:

with (11) package Path:

with (11) package Path:

with (11) package Path:

with (11) package Path:

with (11) package Path:

with (11) package Path:

with (11) package Path:

with (11) package Path:

with (11) package Path:

with (11) package Path:

with (11) package Path:

with (11) package Path:

with (11) package Path:

with (11) package Path:

with (11) package Path:

with (11) package Path:

with (11) package Path:

with (11) package Path:

with (11) package Path:

with (11) package Path:

with (11) package Path:

with (11) package Path:

with (11) package Path:

with (11) package Path:

with (11) package Path:

with (11) package Path:

with (11) package Path:

with (11) package Path:

with (11) package Path:

with (11) package Path:

with (11) package Path:

with (11) package Path:

with (11) package Path:

with (11) package Path:

with (11) package Path:

with (11) package Path:

with (11) package Path:

with (11) package Path:

with (11) package Path:

with (11) package Path:

with (11) package Path:

with (11) package Path:

with (11) package Path:

with (11) package Path:

with (11) package Path:

with (11) package Path:

with (11) package Path:

with (11) package Path:

with (11) package Path:

with (11) package Path:

with (11) package Path:

with (11) package Path:

with (11) package Path:

with (11) package Path:

with (11) package Path:

with (11) package Path:

with (11) package Path:

with (11) package Path:

with (11) package Path:

with (11) package Path:

with (11) package Path:

with (11) package Path:

with (11) package Path:

with (11) package Path:

with (11) package Path:

with (11) package Path:

with (11) package Path:

with (11) package Path:

with (11) package Path:

with (11) package Path:

with (11) package Path:

with (11) package Path:

with (11) package Path:

with (11) package Path:

with (11) package Path:
package body SESSION_MANGER_PKG;

with SESSION_PKG;

package body SESSION_MANGER_PKG is

procedure SESSION_MANGER_PKG :=

begin

end SESSION_MANGER_PKG;
package SNMP-PROCESSOR-OUT 

procedure SNMP-PROCESSOR-OUT (SNMP-MSG-OUT: in MESSAGE; SNMP-OUT: out MESSAGE) is 

package body SNMP-PROCESSOR-OUT is 

procedure SNMP-PROCESSOR-OUT (SNMP-MSG-OUT: in MESSAGE; SNMP-OUT: out MESSAGE) is 

end SNMP-PROCESSOR-OUT; 

end SMP-PROCESSOR-OUT;
and TRK

and TRK

put 'Initialize = Tractol operator TRK]' begin

procedure TRK[TRK : our TRACKS]

package body TRACKS = pkg

package body TRACKS = pkg

package body TRACKS = pkg

procedure TRK[TRK : our TRACKS]

with TRACKS'ExportTRACKS_PKG

select TRACKS_PKG

--purpose: The package processes TRACKS resource
--selected: Thomas L. A. Clark, President of the Science
--location: Cape Mall, Cape Town
--date: 12 June 1997
--author: Cape Mall, Kowall
--language: English

end TIMESTANP
end ENTITY

return dummy

dummy = 0

begin

function ENTITY return TIMESTANP: 
package body TIMESTANP is 
end TIMESTANP;

end ENTITY;

function ENTITY return TIMESTANP: 
package body TIMESTANP is 
end TIMESTANP;

end ENTITY;  

package body TIMESTANP is 

end TIMESTANP;
APPENDIX B

SAAWC MESSAGE SCHEDULE
<table>
<thead>
<tr>
<th>TIME</th>
<th>MASTER</th>
<th>AIR DEF SYS DISPLAY</th>
<th>BROKER</th>
</tr>
</thead>
<tbody>
<tr>
<td>36</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>35</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>34</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>33</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>32</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>31</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>30</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>29</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>28</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>27</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>26</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>25</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>24</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>23</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>22</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>21</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>19</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>18</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>17</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Note: The table seems to be listing various tracks and related functions, possibly related to a computer or server system.
<table>
<thead>
<tr>
<th><strong>Request</strong></th>
<th><strong>Response</strong></th>
<th><strong>Session</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Request key=my secure session</td>
<td></td>
<td>108</td>
</tr>
<tr>
<td>Request correlation=my secure session</td>
<td></td>
<td>107</td>
</tr>
<tr>
<td>valid client req=my secure session</td>
<td></td>
<td>106</td>
</tr>
<tr>
<td>client reg=my secure session</td>
<td></td>
<td>105</td>
</tr>
<tr>
<td>reg signature=my secure session</td>
<td></td>
<td>104</td>
</tr>
<tr>
<td>reg key=my secure session</td>
<td></td>
<td>103</td>
</tr>
<tr>
<td>reg correl=my secure session</td>
<td></td>
<td>102</td>
</tr>
<tr>
<td>reg correl=my secure session</td>
<td></td>
<td>101</td>
</tr>
<tr>
<td>reg correl=my secure session</td>
<td></td>
<td>100</td>
</tr>
<tr>
<td>new time=975096089</td>
<td></td>
<td>99</td>
</tr>
<tr>
<td>print response=the</td>
<td>want to print</td>
<td>98</td>
</tr>
<tr>
<td>valid sy response=the</td>
<td>want to print</td>
<td>97</td>
</tr>
<tr>
<td>sy response=the</td>
<td>want to print</td>
<td>96</td>
</tr>
<tr>
<td>job spec msg=the</td>
<td>want to print</td>
<td>95</td>
</tr>
<tr>
<td>daa stream=the</td>
<td>want to print</td>
<td>94</td>
</tr>
<tr>
<td>receive life=the</td>
<td>want to print</td>
<td>93</td>
</tr>
<tr>
<td>reg data=the</td>
<td>want to print</td>
<td>92</td>
</tr>
<tr>
<td>print data=the</td>
<td>want to print</td>
<td>91</td>
</tr>
<tr>
<td>reg request=the</td>
<td>want to print</td>
<td>90</td>
</tr>
<tr>
<td>res request=the</td>
<td>want to print</td>
<td>89</td>
</tr>
<tr>
<td>valid response=the</td>
<td>want to print</td>
<td>88</td>
</tr>
<tr>
<td>valid client reg=the</td>
<td>want to print</td>
<td>87</td>
</tr>
<tr>
<td>client reg=the</td>
<td>want to print</td>
<td>86</td>
</tr>
<tr>
<td>print file=the</td>
<td>want to print</td>
<td>85</td>
</tr>
<tr>
<td>print file=the</td>
<td>want to print</td>
<td>84</td>
</tr>
<tr>
<td>reg response=the</td>
<td>want to print</td>
<td>83</td>
</tr>
<tr>
<td>reg response=the</td>
<td>want to print</td>
<td>82</td>
</tr>
<tr>
<td>new time=975096089</td>
<td></td>
<td>81</td>
</tr>
<tr>
<td>new time=975096089</td>
<td></td>
<td>80</td>
</tr>
<tr>
<td>new time=975096089</td>
<td></td>
<td>79</td>
</tr>
<tr>
<td>new time=975096089</td>
<td></td>
<td>78</td>
</tr>
<tr>
<td>new time=975096089</td>
<td></td>
<td>77</td>
</tr>
<tr>
<td>new time=975096089</td>
<td></td>
<td>76</td>
</tr>
<tr>
<td>ad del msg=establish</td>
<td>for add</td>
<td>75</td>
</tr>
<tr>
<td>valid sy response=establish</td>
<td>for add</td>
<td>74</td>
</tr>
<tr>
<td>sy response=establish</td>
<td>for add</td>
<td>73</td>
</tr>
<tr>
<td>dev avail new command drive</td>
<td></td>
<td>72</td>
</tr>
<tr>
<td>install device=new command drive</td>
<td></td>
<td>71</td>
</tr>
<tr>
<td>valid device=new command drive</td>
<td></td>
<td>70</td>
</tr>
<tr>
<td>reg del msg=establish for add</td>
<td></td>
<td>69</td>
</tr>
<tr>
<td>valid response=establish for add</td>
<td></td>
<td>68</td>
</tr>
<tr>
<td>print file=establish</td>
<td>want to print</td>
<td>67</td>
</tr>
<tr>
<td>dev avail new command drive</td>
<td></td>
<td>66</td>
</tr>
<tr>
<td>valid device=new command drive</td>
<td></td>
<td>65</td>
</tr>
<tr>
<td>reg del msg=establish for add</td>
<td></td>
<td>64</td>
</tr>
<tr>
<td>valid response=establish for add</td>
<td></td>
<td>63</td>
</tr>
<tr>
<td>print file=establish</td>
<td>want to print</td>
<td>62</td>
</tr>
<tr>
<td>dev avail new command drive</td>
<td></td>
<td>61</td>
</tr>
<tr>
<td>valid device=new command drive</td>
<td></td>
<td>60</td>
</tr>
<tr>
<td>reg del msg=establish for add</td>
<td></td>
<td>59</td>
</tr>
<tr>
<td>valid response=establish for add</td>
<td></td>
<td>58</td>
</tr>
<tr>
<td>print file=establish</td>
<td>want to print</td>
<td>57</td>
</tr>
<tr>
<td>dev avail new command drive</td>
<td></td>
<td>56</td>
</tr>
<tr>
<td>valid device=new command drive</td>
<td></td>
<td>55</td>
</tr>
<tr>
<td>reg del msg=establish for add</td>
<td></td>
<td>54</td>
</tr>
<tr>
<td>valid response=establish for add</td>
<td></td>
<td>53</td>
</tr>
</tbody>
</table>

**TIME**

**BROKER**

**AIR DEF SYS DISPLAY**

**MASTER**
<table>
<thead>
<tr>
<th>TIME</th>
<th>DATA MANAGEMENT</th>
<th>CORRELATION</th>
<th>COMMS</th>
<th>ALERTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.18</td>
<td>1.17</td>
<td>1.16</td>
<td>1.15</td>
<td>1.14</td>
</tr>
<tr>
<td>1.13</td>
<td>1.12</td>
<td>1.11</td>
<td>1.10</td>
<td>1.09</td>
</tr>
</tbody>
</table>

SAWVC MESSAGE SCHEDULE
<table>
<thead>
<tr>
<th>NAME</th>
<th>MESSAGE</th>
<th>MAP</th>
<th>DEVICE</th>
<th>TIME</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>119</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>118</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>117</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>116</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>115</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>114</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>113</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>112</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>111</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>110</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>109</td>
</tr>
</tbody>
</table>
APPENDIX C

ISSUES AND TECHNOLOGIES PERTAINING TO THE DEVELOPMENT OF TRUSTED PATHS IN A DISTRIBUTED HETEROGENEOUS NETWORK
A computing architecture which has come under increasing scrutiny within the computing industry in recent years is that of the three-tier architecture, based on the concept of separating business rules functionality from computational service functionality. Computational services might include infrastructure services such as communication services, data management services, or security services, or they might include common support application services such as correlation services and message processing services. [BUTLER96] proposes for the Department of Defense's (DOD) Global Command Control System (GCCS) a three-tier architecture in which a middle tier acts as a "broker", instantiating client and server "threads" of execution upon requests from clients to "subscribe" to specific services. The broker is responsible for determining not only the appropriateness of client and server subscriptions but also the relative priority of each request. This proposed architecture provides a potential solution to the difficult problem of discriminating between the real-time packet requirements and the non-real-time packet requirements of clients and servers communicating across a single, general purpose network. Though it doesn't describe the protocol for implementing the broker adjudication mechanism, it does identify a separate place for that functionality to reside, simplifying the implementation of both the client and the server functionality by isolating them from the complexity of resolving communication session priorities. For these reasons alone the three-tier architecture is becoming an attractive alternative, not only within the DOD, but also to private sector businesses currently operating within a traditional two-tier client/server architecture.

The nature of this computing architecture, however, is by default, distributed. It is also anticipated that many three-tier implementations would be composed of widely varying computing
platforms, due to the economic pressures associated with leveraging investments in legacy systems and the scaled upgrading of individual systems within a network architecture. The combination of distributed computing and heterogeneous platforms within a three-tier architecture, however, poses a significant challenge to the development of secure computing mechanisms which can provide the degree of trust required by locally developed security policies. In order to meet this challenge, new security technologies designed for inter-network environments, and new methods for implementing legacy security technologies must be investigated as alternatives to the numerous proprietary intra-network security mechanisms in widespread practice today.

From [CSC97], the following scenario provides motivation for why a typical GCCS operator might require the use of a trusted path between platforms and across a distributed processing, heterogeneous network in the course of conducting an operational task:

"...The user inserts his or her Fortezza card into the work station card reader, authenticates against the card and then downloads an Applet and begins its execution. The Applet communicates with a server using either CORBA method invocations or DCE remote procedure calls. The Applet and/or the server with which it is communicating specify via CORBA/DCE APIs the amount and type of security to invoke: authenticate the user (or the application server), verify that transmitted data has not been modified, completely encrypt all communication, etc. CORBA or DCE, in turn, utilizes the Fortezza and X.500 APIs to perform the user authentication, encryption, etc. Thus, Java builds its interprocess communication on top of either CORBA or DCE, which in turn builds its security on top of Fortezza and X.500."

From the preceding paragraph one can infer a strong commitment on behalf of the DOD's Defense Information Systems Agency (DISA) toward two trends: distributed computing in a
heterogeneous environment, and the use of legacy security mechanisms to implement local security policies. [CSC97] describes the four cornerstones of a distributed security infrastructure for the network-centric GCCS: Fortezza, X.500, CORBA, DCE. Fortezza is a standard for public/private key cryptography and is available as a PC card implementation for those platforms supporting this mechanism. X.500 is an International Standards Organization (ISO) standard which specifies a global, hierarchical name service implemented as a distributed database accessible via Lightweight Directory Access Protocol (LDAP) clients or by applications using the X.500 client APIs. Entries contained within an X.500 directory are treated as objects, fully configurable and consisting of a collection of attributes. In the GCCS infrastructure, X.500 will serve two purposes: as the repository for public key storage in the Fortezza scheme, and as a white pages for the Defense Message System (DMS), the DOD's projected replacement for the legacy AUTODIN messaging system. The Object Management Group's (OMG) Common Object Request Broker Architecture (CORBA) is a proposed architecture for the creation and interaction of distributed objects, and the Distributed Computing Environment (DCE) is a comparable architecture standard proposed by the Open Software Foundation (OSF). A deliberate investigation into the complexity of trusted paths, and the mechanisms required to realize them, illustrates why the previously described technologies are just a few of many potential solutions to the problem of secure computing in a distributed, heterogeneous environment.

The fundamental requirement for two processes to communicate securely in a computing environment is encapsulated in the concept of a trusted path. At the simplest level, a process currently running in the Central Processing Unit (CPU) might communicate directly with a
process driving an Input/Output device, such as a keyboard or harddrive, via an interrupt-driven
data transfer mechanism known as a bus, or with another embedded processor such as might be
found on a video card or modem. At the most complex level, communicating processes might be
executing on physically distinct machines separated by thousands of miles and subjected to the
protocol packaging and unpackaging of network processors responsible for routing data between
the two processes. The establishment of a trusted path between two communicating processes
certainly becomes more difficult to achieve as the number of required cooperating and assisting
processes responsible for that communication increases. Intuitively, it becomes much more of a
challenge to anticipate the numerous "portals" for attacking data transiting over miles and
between machines than it is to plan for the protection of data transiting over inches and within the
confines of a single machine. And yet the computing patterns of today point to an ever-increasing
need to connect more computing machines together via networks and inter-networks that are
increasingly subject to malicious attacks. In order to combat these attacks, current trends in
computing point toward the use of computer-derived manifestations of long trusted human
mechanisms, such as trusted paths and secure domains of operation, through the use of
certificates of trust, encryption of data, and signatures as the means for guaranteeing, respectively,
authenticity, secrecy, and integrity of the data transiting between two communicating processes.

Demonstrating a mechanism for the establishment of a local trusted path, the
[GARFINKEL96] text describes a procedure between the operating system and the login program
which is invoked via signals from the keyboard receiving its input from the human operator.
Specifically, the operating system employs a mechanism to kill all running processes upon receipt
of a particular signal from the keyboard. This policy ensures that the only running processes are those legitimate processes associated with the real operating system. What makes this type of trusted path mechanism possible is the presence of both a direct path between devices and a proprietary addressing scheme, i.e., MCA, ISA, IRQ, etc., to foil the attempt of any malicious process attempting to masquerade as a legitimate process.

But establishing direct and trusted paths between processes identified by non-standard, close-ended addressing schemes, such as the mentioned bus schemes, is neither practical nor even interesting, given the scale in which people expect to compute and communicate today. In order to close the physical distance between human and computing activities we've achieved the affect of logical collocation through the implementation of standardized datalink, network, and inter-network-level computing protocols. And though we've significantly complicated the task of establishing trusted paths between communicating processes, our need for trusted paths has only increased: the speed at which we've connected together our computing devices is exceeded only by the speed at which we've devised ways to automate so many human activities. Further complicating the task of creating and sustaining trusted paths is the shift in thinking about the ways in which we program: from procedural to object-oriented, from writing code to automate the conduct of a transaction to writing code which simulates two or more transacting objects. So not only are we concerned with guaranteeing the authenticity, secrecy, and integrity of transiting data, but also with programming transiting binary globs of intelligence which can alter the way in which our two (or more) processes are communicating to more closely mirror the real world events we seek to automate. But if the future of computing points toward an ever-increasing
need to communicate over inter-networks, as client objects, server objects, and middle-tier broker objects, and within the new paradigm of object-oriented programming which shifts focus from programming the procedural to programming the behavior and the interactions of objects, then we need to identify the means with which we can provide the authenticity, secrecy, and integrity that communicating processes require to establish a trusted path in this new environment. Perhaps the widely endorsed, object-oriented standard middleware architecture, CORBA, and the rapidly growing, object-oriented programming language, Java, are the best tools currently available to developers to achieve those means. Furthermore, there exist projects, such as Trusted Information Systems' (TIS) SIGMA, which have examined the capabilities of these development and runtime environments and derived potential solutions to the problem of trusted communication between objects over an inter-network.

There are two principal advantages to the developer using Java and CORBA tools together to develop for a distributed, heterogeneous computing environment composed of an array of different computer architectures, operating systems, and network protocols. The CORBA specification provides an environment and a standard for writing and reading object interfaces. These allow the user to preserve his investment in legacy code by "wrapping" these legacy "objects" in a standard Interface Definition Language (IDL) interface, which is then propagated across the Object Request Broker, or "ORB" bus, so that client objects can recognize and invoke the published methods of the server objects. The Java standard, on the other hand, provides a tool with which the developer can create, in any Java development environment, the implementation for either the client or server objects, or both, with the result that the compiled

269
Java "bytecode" is portable across any computing platform running the language standard Java Virtual Machine (JVM). This makes it an attractive alternative to writing and compiling the same objects on each of many desired platforms. What is compelling about the capabilities inherent in these two software development standards is that users can take advantage of existing legacy authentication mechanisms such as Kerberos servers and encryption mechanisms such as Fortezza and concentrate on building secure clients and middle-tier brokers guaranteed to run on any platform running a JVM. [ORFALI97] describes CORBA as bringing to distributed computing "network transparency" while Java brings to heterogeneous computing "implementation transparency."

A CORBA/Java architecture is designed to provide the infrastructure for moving objects across a Transmission Control Protocol/Internet Protocol (TCP/IP) network and for enabling communication between those objects. It is proposed as an alternative architecture to that provided by existing TCP/IP distributed processing protocols such as Common Gateway Interface/HyperText Transfer Protocol (CGI/HTTP), JavaSoft's Remote Method Invocation (RMI), and Microsoft's Distributed Component Object Model (DCOM). The CGI/HTTP architecture is currently the most popular model for providing the means for portable clients (generally a JVM running in a Web Browser) to access data from legacy database servers. While this functionality provides a significant leap over previous proprietary architectures for client/server distributed computing, it does not address the need to build client and server objects which can determine requirements and request services dynamically, at runtime. RMI and DCOM, on the other hand, do provide the architecture for distributed computing with objects.
However, they are proprietary solutions with acceptance limited to specific targeted communities. CORBA is the standard developed by the OMG which has been embraced by over 700 computing industry companies. Likewise, the Java language is being widely embraced for its highly recognizable syntax (C, C++ like), its strong software engineering attributes (Ada95 like), its strong object-oriented nature (Smalltalk like), and its support for key distributed object-oriented computing concepts such as multiple threads of execution and its built-in networking Application Programming Interfaces (APIs). Finally, a key component of the CORBA/Java architecture is the increasing acceptance and deployment in commercial products of the Internet Inter-ORB Protocol (IIOP), a principal TCP/IP protocol standard for managing communication between objects and different ORBs on a TCP/IP network, discussed in greater detail later.

The CORBA specification is purposefully neutral with regard to the programming language of choice for the implementation of security policy objects, or other CORBA objects for that matter. Java is a leading candidate for several reasons: the Java language specification brings to the table a set of language-specific mechanisms for achieving object-oriented programming, multi-threaded performance, and dynamic deallocation of memory no longer in use; it also brings a runtime environment with specific rules for trusted access to system resources. Given the growing popularity of the JVM runtime environment as a host environment for the execution of network distributed mobile code, and anticipating a day when JVM-hosted applets might be performing the preponderance of computation at the client level, an investigation into the security policy and mechanisms of the JVM is highly relevant. Because even the most robust authentication mechanisms, encryption algorithms, and integrity-checking schemes are trivial
obstacles to a malicious individual with the capability to manipulate the runtime environment itself.

The JVM presents a seemingly uniform interface to the Java programmer which permits him to concentrate on writing classes which may invoke (and assume the automatic invocation) of the targeted JVM security mechanisms, regardless of the platform that JVM is running on. In reality, however, each JVM is in fact, platform-specific. JVM vendors are given great leeway to pick, choose, and implement the type and degree of security mechanisms identified in the JVM specification. This flexibility allows for Applet viewers (JVMs) to be written for inherently more trusted environments, such as a corporate intranet or secure Local Area Network (LAN), as well as for untrusted environments, such as a general-purpose, commercially distributed Web browser might be used in. The specification identifies two layers of defense: the first is a mechanism for generating and validating digital signatures; the second is a policy, the "sandbox," which defines the system resources which are within and off-limits to the executable mobile code. Applets with validated signatures are permitted the same execution privileges as locally stored application code. Locally stored application code, though not guaranteed to be free of malicious code, is assumed to be; the reality being that any executable code, regardless of origin, must ultimately be given the green light or rejected by a user at some point in time, based upon local procedures for obtaining and loading that code to local storage. The JVM uses three mechanisms to implement its sandbox policy: the Bytecode Verifier, the Class Loader, and the Security Manager. [FLANAGAN96] lists the following privileges unavailable to an unauthorized Applet: reads, writes, deletions, and renaming of files; creating, removing, and viewing directories; searching for a file or reading a
file's attributes; creating network connections to any computer other than the originating host; listening for or accepting network connections on any local ports; creating a top-level window without indicating that the window is "untrusted"; obtaining current user name or home directory; defining system properties; running other programs locally; causing the Java Interpreter to exit; loading dynamic libraries locally; creating or manipulating threads that are not part of the Applet's ThreadGroup; manipulating any ThreadGroup other than its own; creating a ClassLoader or Security Manager object; specifying network control classes; accessing or loading classes in any package not in the standard API eight; or defining classes that are part of packages on the local system.

The first of the JVM security mechanisms, the Bytecode Verifier, has the responsibility for checking downloaded executable code for namespace or type conversion violations. [FLANAGAN96] notes that the Bytecode Verifier ensures that the code is valid JVM code, neither overflows nor underflows the stack, does not use registers incorrectly, and does not convert data types illegally. The principle concern for the Bytecode Verifier is that malicious code might forge pointers or use memory arithmetic to escape the "sandbox" and gain access to regions of memory assigned to other applications or to the operating system. An additional concern to the Bytecode Verifier is that malicious code might cause the Java Interpreter to become unstable and take advantage of resulting or previously existing security holes. The responsibility of the ClassLoader is to dictate the runtime environment by installing each Applet in its own namespace and by prohibiting Applets from seeing and referencing classes from outside of its namespace. This denies a malicious Applet the opportunity to replace the Java API class

273
libraries with its own versions. Finally, the Java Security Manager mechanism consists of a collection of mechanisms, or methods, which can be used by the system to verify whether or not certain operations are allowed in the current runtime environment. It is the Security Manager object instantiated by a particular Applet viewer which enforces the security policy specified by that JVM.

Though the Java language specification is inherently network and distributed computing oriented, it still must rely upon some lower level protocol to enable the movement of those distributed objects. Distributed computing which could be achieved on a large-scale and in a non-proprietary fashion arrived in the mid-1980s with the work of Sun Microsystems and UC Berkeley on the Remote Procedure Call (RPC) protocol. This established a mechanism for allowing a process to invoke the computational functionality of a remote machine across a TCP/IP network. The principle advantage that invoking object methods via a CORBA ORB has over more traditional methods of invoking functions across a network, such as RPC, is that, with RPC, the called function has no state. The calling object is invoking a statically determined function with statically determined data sets. With CORBA, on the other hand, the calling object is invoking a specific function (method) of an object which has state, and the results of the call are dependent upon the dynamically determined condition of the called object's data sets (attributes) at the time of the call. In other words, the polymorphic behavior that we have come to expect in local computations in programs written in languages which support that behavior, can now be realized across a network in a distributed, heterogeneous computing environment. Specifically, [ORFALI97] describes how a CORBA ORB provides for either the statically defined or the
dynamically discovered invocation of remote object methods, language-neutral data types, runtime tables of data (metadata) which allow client objects to dynamically discover the methods of server objects, and transparency to the programmer with regard to issues of transport, client and server location, object activation, and byte ordering.

In addition to the ORB, which dictates the mechanism for invoking the methods of remote objects, the most recently adopted CORBA specification, CORBA 2.0 approved in 1995, defines a set of 16 CORBA system-level services which define the means for creating, naming, copying, moving, deleting, registering, locking, relating, and publishing objects and information about those objects. These services also include rules for committing on transactions between objects, a superset of Structured Query Language (SQL) operations to support access to Database Management Systems (DBMS), a licensing service to support fair and mediated access to certain objects, a time service to synchronize interactions between objects, and a security service, defined below in greater detail, which specifies rules for authentication, access control lists, confidentiality, and non-repudiation. [ORFALI97] describes the advantages of developing in a CORBA environment by using the example of developing a car. The developer can create a car "component" by inheriting concurrency, persistence, and transaction awareness from the defined CORBA services. Similarly, a developer could create security policy objects which simplify the means by which security policy is created, published, and enforced across a heterogeneous network.

The CORBA security environment defined in the CORBA 2.0 specification describes a security model and architecture, and it leaves the selection of security mechanisms to apply to that
model up to the implementor. Possible mechanisms for employment in a CORBA security model include Kerberos, Secure RPC, and Secure European System for Applications in a Multivendor Environment (SESAME). The principle motive of the people responsible for developing this specification was to decouple the implementation of security mechanisms from the implementation of client and server processes and applications. In other words, developers would be free to implement security mechanisms into their client and server objects, but could also expect to receive the protections offered by authentication, access controls, encryption, signing, and auditing which can be designed into the ORB itself by the ORB vendors. As a means to achieving these security protections, it is the intention of the specification authors that developers of CORBA security mechanisms will provide for a Credentials object, created when a user logs in or a process is invoked, which will contain the user's/processes' privileges regarding roles, groups, and security clearance. Objects then invoked by the ORB will access the Credentials object as a first step in determining the identity and privileges of the invoking object.

A recurring theme in the discussions of secure computing within a heterogeneous, distributed computing architecture is the notion of a trusted intra-network, or secure domain. In their SIGMA project, TIS speaks of these domains as "enclaves." The project managers selected the term enclave to describe a network environment which retains interoperability with other networks but is nevertheless protected from those outside networks through locally established security policies. The [GARFINKEL96] text presumes that the nature of trust within the enclaves themselves is assured through traditional measures such as good hiring practices, good account administration, good password assignment, and good physical security of the local area network.
For this reason the text is primarily devoted to the discussion of inter-network security. In military computing, where existing personnel and weaponry form a strong barrier to potential physical threats to a network, and where a rigid policy regarding the recruitment of candidates and the training of operators forms a significant first layer of defense, we make the same assumptions regarding the security of our own enclaves with the result that we, too, are increasingly concerned with the inter-enclave threats against which our enclaves are most vulnerable. [GARFINKEL96] describes in great detail the policies and mechanisms available to protect these trusted enclaves from networks external to them through a combination of chokes and gateways which, when properly configured, constitute a firewall; as well as through simpler, process-independent mechanisms such as wrappers. Both of these constitute aggressive mechanisms for the rigid filtering of suspicious IP packets and for the use of proxy processes which handle internal and external requests for service. But current firewall mechanisms are incapable of completely addressing the complex security needs of objects which are interacting across the inter-network. It is the interconnection between these trusted and untrusted enclaves, the gateway and the policies and mechanisms which compose the gateway, which is the target of efforts to combat threats to the network and is the focus of effort in the SIGMA project.

TIS's SIGMA project is a research effort developed to investigate and prototype a collection of security mechanisms which implement domain-specific security policies between trusted and untrusted systems across a heterogeneous distributed computing environment based on CORBA interoperability. From "http://www.tis.com/docs/research/distributed/sigma.html", the purpose of SIGMA is threefold:
Develop security mechanisms for protecting an enclave by controlling access by other enclaves with which it interoperates.

Improve the state of the art of security mechanisms for object-oriented distributed systems.

Extend interoperability access controls to apply to heterogeneous security mechanisms and disparate policies of different enclaves.

The SIGMA project recognizes and addresses the need for communication between three distinct enclave types: a Multi-Level System (MLS) enclave in which information is controlled by strong label-based separation mechanisms; a Domain and Type Enforcement (DTE) enclave in which information is subject to complex role-based policies; and a Commercial Off The Shelf (COTS) enclave, in which information is subject to unknown or untrusted security policies. The project further recognizes that even among like enclave types, there will be differences in security policies, security mechanisms, and levels of assurance. [BENZEL96] documents an analysis of security policies and mechanisms in the current CORBA security specification and concludes the following topics are not adequately addressed: required security functionality for interoperability between enclaves and high assurance mechanisms for interoperability within enclaves.

[BENZEL96] identifies one significant obstacle facing the CORBA security object developer as being the large-scale deficiency in common commercially sold Operating Systems to guarantee that developed security components cannot be bypassed or tampered with. Contributing to this concern, the authors of the study have concluded that, due to performance concerns, ORB implementations largely consist of library modules residing in the same process address space as the client and server object processes they are designed to support. This
prevents the establishment of independent security mechanisms which can run, monitor, and interrupt questionable transactions.

As mentioned previously, the question of inter-enclave security is one which can be considered outside of the context of the given security policies and mechanisms of the enclave itself. The authors in [BENZEL96] stress that the single point of control for a network that is the network gateway is not so much a design strategy as much as a consequence of network architecture. This consequence presents an opportunity for the network security planner to perhaps compensate for the security weaknesses inherent in his deployed Operating Systems: whatever high-assurance security mechanisms may be absent in local OS's can be deployed at the network gateway in the form of choke mechanisms which provide IP packet-level monitoring and discarding, and in the form of gate mechanisms which provide proxy applications for accomplishing remote processing. Any security mechanisms implemented at the OS, application, or ORB level within the enclave only complement those established at the network gateway and constitute part of a "defense in depth" security policy and a prudent means for dictating the practice of secure computing within the enclave.

In SIGMA project terminology, the single point of access control for the network in question is called the ORB Gateway. Like a network firewall, the ORB Gateway consists of a set of security mechanisms which examine incoming and outgoing data to determine its compliance with the network security policy. Unlike a firewall, though, the ORB Gateway performs its functions by interrogating, authenticating, and validating objects and object requests before permitting their movement into and out of the enclave. The methodology for implementing the
ORB Gateway security mechanisms is the concept of DTE. It is through the distinct DTE signature of each application service, method, object, object attribute, and invoking object attribute that the security mechanisms of the ORB Gateway are able to establish the degree of trust specified by the network security policy.

A principle concern of CORBA ORB and CORBA object developers, and of the SIGMA project, is the movement of objects, security-related or otherwise, through that single point of access and across the distributed, heterogeneous network. In an environment in which competing, distributed, object-oriented computing paradigms abound, there is a requirement for a TCP/IP-based protocol which enables communication between objects managed by different ORBs. That protocol is the Internet Inter-ORB Protocol (IIOP), and it represents the common language between different ORBs which permits one ORB to correctly interpret the requests and responses of another, dissimilar ORB. IIOP makes the assumption that it is using a connection-oriented TCP session and it specifies to each ORB the acceptable data representation and message formatting. This is accomplished through Common Data Representation (CDR) coding, which defines a coding for all Interface Definition Language (IDL) data types, including primitive types, structured types, and object references. However, since most current firewalls do not support the capability to identify and service IIOP packets, the SIGMA project seeks to configure a traditional network firewall which sends IIOP packets to an ORB Gateway configured to handle only IIOP traffic. Incorporating a firewall feature known as a "plug", the firewall can be configured to forward to the ORB Gateway all IIOP traffic received on a particular TCP port. An additional reason for isolating IIOP packet processing from the network firewall is the complexity
of that processing, which contradicts the goal of incorporating small, simple, well-documented functionality within the firewall to filter out suspicious data packets.

[BENZEL96] identifies three forms of restriction performed by the ORB Gateway which demonstrate how an ORB Gateway matches and exceeds the capability for secure distributed computing that is provided by traditional firewalls. The first restriction is on the enclave-resident CORBA-based application services available to outside users. The CORBA model provides for both the static and dynamic discovery of application services through the publishing of interface definitions in a common IDL. The ORB Gateway will be configured with information pertaining to the enclave application services, including which services are accessible to outside users and over which communication ports those services may be requested. The second restriction is on the specific methods which can be invoked by an outsider; perhaps a subset of those methods offered by the CORBA server object. Reasons for restricting access to certain methods might be to provide read-only or write-only access to a particular server object in order to dictate in what way client objects may and may not alter the state of the server object. The enforcement of method restriction is controlled by the composition of a configuration list of methods used by the ORB Gateway to implement the enclave security policy. A third restriction encapsulates what is unique about CORBA and object-oriented programming: the restriction of specific objects to requesting objects without regard to the presence or absence of restrictions specified for access to methods or services as mentioned above.

[BENZEL96] identifies two approaches for an ORB Gateway using authentication data to restrict outside access to enclave CORBA objects. In the first approach, access checks are
performed by the ORB Gateway based upon a reconciliation between the contents of locally held configuration data and the object request message data. Authentication of the object request is handled internally, allowing the security administrator to develop separate and distinct security policies regarding access control for users within and outside of his enclave. In the second approach, access checks are not performed. Instead, the authentication mechanisms and data of the foreign enclave are translated into the comparable authentication mechanisms and data of the local enclave. With the bundled authentication data, or "credentials", the object request can be passed into the enclave where the ORB can make the appropriate access control decisions just as if the request had been initiated from within the enclave itself. This approach has the advantage that all access control decisions to an enclave object can be made at a central point, perhaps simplifying the administration of mechanisms such as access control lists.

Together, the Java language specification, the CORBA specification, and TIS's SIGMA project represent three new technologies which can significantly enhance the establishment of the trusted paths between communicating processes which GCCS operators require in a distributed, heterogeneous network. TIS's approach to establishing trusted paths between enclaves of computer networks is noteworthy for two reasons: its recognition that objects present the most promising method for encapsulating the security credentials of a given person or process, and its adherence to an open architecture which permits interoperability between competing ORB implementations and between enclaves with significantly varying internal security policies. The CORBA standard provides a framework both for supporting the interoperability between different proprietary object implementations and for defining an environment in which legacy
implementations can interact with fully object-oriented implementations. Finally, the Java language specification provides the basis for multi-threaded processing, platform-independent Graphical User Interface (GUI) construction, and built-in support for inter-networking which permits development of the security mechanisms prescribed by CORBA and SIGMA and are necessary for the establishment of trusted paths between users and processes in a distributed, heterogeneous computing environment.
LIST OF REFERENCES


# INITIAL DISTRIBUTION LIST

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
<th>Number of Copies</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Defense Technical Information Center</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>8725 John J. Kingman Road, Ste 0944</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Ft. Belvoir, VA 22060-6218</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Dudley Knox Library</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>Naval Postgraduate School</td>
<td></td>
</tr>
<tr>
<td></td>
<td>411 Dyer Rd.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Monterey, California 93943-5101</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>ECJ6-NP</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>HQUSEUCOM</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Unit 30400 Box 1000</td>
<td></td>
</tr>
<tr>
<td></td>
<td>APO, AE 09128</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Director, Training and Education</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>MCCDC, Code C46</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1019 Elliot Road</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Quantico, VA 22134-5027</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Director, Marine Corps Research Center</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>MCCDC, Code C40RC</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2040 Broadway Street</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Quantico, VA 22134-5107</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Director, Studies and Analysis Division</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>MCCDC, Code C45</td>
<td></td>
</tr>
<tr>
<td></td>
<td>3300 Russell Road</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Quantico, VA 22134-5130</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>Marine Corps Representative</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Naval Postgraduate School</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Code 037, Bldg. 234, HA-220</td>
<td></td>
</tr>
<tr>
<td></td>
<td>699 Dyer Road</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Monterey, CA 93940</td>
<td></td>
</tr>
</tbody>
</table>
8. Marine Corps Tactical Systems Support Activity ................................. 1
   Technical Advisory Branch
   Attn: Maj J.C. Cummiskey
   Box 555171
   Camp Pendleton, CA 92055-5080

9. Dr. Luqi, Code CS/Lq........................................................................ 2
   Department of Computer Science
   Naval Postgraduate School
   Monterey, California 93943-5002

10. Dr. Man-Tak Shing, Code CS/Sh.................................................... 1
    Department of Computer Science
    Naval Postgraduate School
    Monterey, California 93943-5002

11. CDR Michael J. Holden USN, Code CS/Hm........................................ 1
    Department of Computer Science
    Naval Postgraduate School
    Monterey, California 93943-5002

12. Dr. Ted Lewis, Code CS/Le............................................................ 1
    Department of Computer Science
    Naval Postgraduate School
    Monterey, California 93943-5002

    Department of Computer Science
    Naval Postgraduate School
    Monterey, California 93943-5002

14. Curricular Officer, Code 32........................................................... 1
    Naval Postgraduate School
    Monterey, California 93943-5002

15. Capt Matthew P. Howell USMC...................................................... 2
    20966 Promontory Sq.
    Sterling, VA 20165-7211