NAVAL
POSTGRADUATE
SCHOOL

MONTEREY, CALIFORNIA

THESIS

INTEGRATING DISTRIBUTED INTERACTIVE SIMULATIONS WITH THE PROJECT DARKSTAR OPEN-SOURCE MASSIVELY MULTIPLAYER ONLINE GAME (MMOG) MIDDLEWARE

by

Tariq Rashid

September 2009

Thesis Advisor:	Don Brutzman
Thesis Co-Advisor:	Don McGregor
Second- Reader:	Amela Sadagic

This Thesis Done in Cooperation with the MOVES Institute

REPORT DOCUMENTATION PAGE			*Form Approved OMB No. 0704-0188*
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.			
1. AGENCY USE ONLY ***(Leave blank)***	**2. REPORT DATE** September 2009		**3. REPORT TYPE AND DATES COVERED** Master's Thesis
4. TITLE AND SUBTITLE: Integrating Distributed Interactive Simulations with the Project Darkstar Open-Source Massively Multiplayer Online Game (MMOG) Middleware			**5. FUNDING NUMBERS**
6. AUTHOR: Tariq M. Rashid			
7. PERFORMING ORGANIZATION NAME AND ADDRESS Naval Postgraduate School Monterey, CA 93943-5000			**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING /MONITORING AGENCY NAME AND ADDRESS Office of Naval Research , One Liberty Center 875 North Randolph Street, Suite 1425. Arlington, VA 22203-1995			**10. SPONSORING/MONITORING AGENCY REPORT NUMBER**
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.			
12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited			**12b. DISTRIBUTION CODE**

13. ABSTRACT

Recently, a great deal of attention has been given to the use of Massively Multiplayer Online Games (MMOGS) for both gaming and military applications. The revenue generated by MMOGs and the effect that they have on the network infrastructure has resulted in significantly more developmental resources being applied to commercial MMOG technology than for military distributed virtual (DVE) development. All DVEs share a common set of characteristics, and additional requirements exist for the interoperability of military DVEs. It is possible to exploit these similarities to take advantage of developments in the supporting technologies of commercial MMOGs.

Specific capabilities of interest include scalability for large numbers of players, capacity for large amounts of network traffic, portability across operating systems, and adaptability to connect diverse codebases, network protocols, and data formats. Project Darkstar is a Sun Labs research project, which has developed an open-source middleware for MMOGs. This thesis has produced and tests a MMOG server, which interconnects heterogeneous simulators in a DVE using the Project Darkstar middleware and locally developed network gateways. The performance of the system and the character of the network traffic it generates are analyzed. Initial test results warrant further development and eventual deployment.

14. SUBJECT TERMS Distributed Interactive Simulation, Massively Multiplayer Online Game, Simulation Interoperability			**15. NUMBER OF PAGES** 136
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT Unclassified	**18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	**19. SECURITY CLASSIFICATION OF ABSTRACT** Unclassified	**20. LIMITATION OF ABSTRACT** UU

NSN 7540-01-280-5500 Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

INTEGRATING DISTRIBUTED INTERACTIVE SIMULATIONS WITH THE PROJECT DARKSTAR OPEN-SOURCE MASSIVELY MULTIPLAYER ONLINE GAME (MMOG) MIDDLEWARE

Tariq M. Rashid
Lieutenant Commander, United States Navy
B.S., Georgia State University, 1994

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN MODELING VIRTUAL ENVIRONMENTS AND SIMULATION (MOVES)

from the

NAVAL POSTGRADUATE SCHOOL
September 2009

Author:	Tariq M. Rashid
Approved by:	Don Brutzman Thesis Advisor
	Don McGregor Co-Advisor
	Amela Sadagic Second Reader
	Mathias Kölsch Chair, MOVES Academic Committee

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

Recently, a great deal of attention has been given to the use of Massively Multiplayer Online Games (MMOGS) for both gaming and military applications. The revenue generated by MMOGs and the effect that they have on the network infrastructure has resulted in significantly more developmental resources being applied to commercial MMOG technology than for military distributed virtual (DVE) development. All DVEs share a common set of characteristics, and additional requirements exist for the interoperability of military DVEs. It is possible to exploit these similarities to take advantage of developments in the supporting technologies of commercial MMOGs.

Specific capabilities of interest include scalability for large numbers of players, capacity for large amounts of network traffic, portability across operating systems, and adaptability to connect diverse codebases, network protocols, and data formats. Project Darkstar is a Sun Labs research project that has developed an open-source middleware for MMOGs. This thesis has produced and tests a MMOG server, which interconnects heterogeneous simulators in a DVE using the Project Darkstar middleware and locally developed network gateways. The performance of the system and the character of the network traffic it generates are analyzed. Initial test results warrant further development and eventual deployment.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF ACRONYMS AND ABBREVIATIONS

AOIM	Area-of-Interest Management
AUV	Autonomous Unmanned Vehicle
C4I	Command, Control, Computing, Communication and Intelligence
DIS	Distributed Interactive Simulation (Specifically IEEE-1278)
DMO	Distributed Mission Operations
DoD	Department of Defense
DVE	Distributed Virtual Environment
ESPDU	Entity State Protocol Data Unit
FOM	Federation Object Model
FPS	First-Person Shooter
HLA	High Level Architecture (Specifically IEEE 1516)
IER	Information-Exchange Requirements
IETF	Internet Engineering Task Force
IGDA	International Game Developers Association
KB	Kilobyte
Kbps	Kilobits per Second
LAN	Local-Area Network
LESS	Live Entity State Stream
M&S	Modeling and Simulation
Mbps	Megabits per Second
MMO	Massively Multiplayer Online
MMOG	Massively Multiplayer Online Game
MMORPG	Massively Multiplayer Online Role-Playing Game
MMRTS	Massively Multiplayer Real-Time Strategy
MRT3	Mission Rehearsal Tactical Team Trainer
ms	millisecond (s)
MTC	Mission Training Center
MUD	Multi-User Dungeon
PDU	Protocol Data Unit
QoS	Quality-of-Service

RPG	Role-Playing Game
RTI	Run-Time Infrastructure
SIMNET	Simulator Networking
SMAL	Savage Modeling and Analysis Language
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
WAN	Wide-Area Network
XML	Extensible Markup Language

ACKNOWLEDGMENTS

I would like to first thank my wife, Princess, for her understanding and support while I spent many an evening in academic pursuits. I hope to be as supportive of her present and future endeavors as she is of mine.

I would like to thank the Faculty and Staff of the Modeling, Virtual Environment and Simulation (MOVES) Institute and the Naval Postgraduate School for a challenging and interesting curriculum. I would not have come here to do something easy and it certainly was not. My life has changed because of what I have learned here. I would especially like to thank Don Brutzman, Don McGregor, and Amela Sadagic for their indispensible guidance, advice, and help on my thesis.

Completion of my thesis would have likely been impossible without the sponsorship of the Office of Naval Research. Specifically Garth Jensen’s interest and support has been critical. Douglas Maxwell at the Naval Undersea Warfare Center also provided important support.

Lastly, I would like to thank Gordon Nakagawa (CAPT, USN Ret) for the opportunity to have been captain of the NPS Marksmanship team under his tutelage. What I learned in that role is as much a part of my education here at NPS as any other single element.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

A. PROBLEM AND THESIS STATEMENT

Throughout the history of games, there has been interplay between gaming for military purposes and gaming for entertainment purposes (Smith, 2009; Zyda, 1997). Attention has lately been directed towards finding military applications for Massively Multiplayer Online Games (MMOG) (Bonk & Dennen, 2005). MMOGs continue to be one of the fastest growing segments of the computer game market. Typically, the commercial MMOG environment is dominated by role-playing games (RPG). This is a function of the economics of commercial MMOG development, sales and operation. There is nothing inherent in MMOG architecture that requires them to be RPGs. The inherent characteristics of MMOGs are scalability, persistence, and a consistent world state.

For purposes of this thesis, the term Distributed Virtual Environment (DVE) is used to mean a scalable, virtual environment that enforces to some degree a common world state for all clients independent of a specific session. Several important questions relate to use of MMOG technology in military simulations. What state information needs to be made available to participants in this environment? Is it possible to leverage the investment in existing simulators by adapting them to function as the user-interface for a persistent virtual environment? Can improved performance or additional functionality be gained by using Distributed Interactive Simulation (DIS) protocol and MMOG hybrid architecture? These questions are examined in this thesis.

B. MOTIVATION

The military and business communities have long noted the potential for simulation and gaming technology to develop high-order thinking skills. It has been suggested that certain skills gained and practiced by gamers in multiplayer online gaming environments closely parallel those required by teams of operators interacting in the real world. This analogy might be pushed even further to that of a military transforming to operate under the concept of network-centric warfare. A DVE that is able to maintain an objective state independent of whether any participants are connected to it may lend itself to medium-term and long-term virtual exercises. Sufficient scalability is needed to allow large numbers of participants. The growth and popularity may mean that technology to facilitate development and operation of DVE for military applications might be made rapidly available by taking advantage of the competition within the gaming industry.

It is important to note that a DVE and an MMOG are not necessarily the same. By way of distinction, a game may be described as a real-time system having actors, goals, rules, feedback in the form of a score, and some sort of plot or scenario. A DVE as described here may only possess some of these characteristics. A DVE intended to be used for training and experimentation may only need to support hundreds of participants, not the tens or hundreds of thousands of linked players that commercial MMOGs must support in order to be economically viable. Given the scope of today’s problems of interest, however, hundreds of participants are hardly “Massive.” Explicit user interaction within a DVE is usually defined at the client level by the application interface. Such interaction can range from text messages from a mobile phone to immersive stereo and data gloves in a 3D virtual world.

Currently, the IEEE Distributed Interactive Simulation (DIS) network protocol is widely used to enable interoperability for distributed simulation in military applications. In order to deliberately move from known to projected DVE capabilities, this thesis work first takes advantage of the rich set of DIS-enabled X3D models provided by the online Scenario Authoring and Visualization for Advanced Graphical Environments (Savage) and For Official Use Only (FOUO) SavageDefense model archives. The ongoing production of agent-based and robot-based models that are completely visualized using X3D provides a large and growing set of exemplars. Prior thesis work comparing prominent DVEs (Sanders, 2008) provides a good summary of baseline capabilities. Relevant thesis work regarding metadata for distinguishing entities is SAVAGE Modeling and Analysis Language (SMAL) (Rausch, 2006).

It is also possible to develop a local DIS interface to the SH-60B MRT3 training device and other simulations that enable communication with a persistent virtual environment. Such an interface is developed, described and demonstrated in this thesis. This proof of concept demonstrates that any real-time training or tactical device, which uses DIS for interoperability, is also able to participate in the DVE. Additional data formats and network protocols with semantically similar information can be then as added as connections to this MMOG server.

C. SCOPE OF THE THESIS

The thesis is limited to an exploration of the requirements for a DVE to support distributed training and experimentation. This requirements set is used to develop a recommended architecture for such a DVE. A small-scale DVE is developed using Project Darkstar® as middleware. Lessons learned from this demonstration are then used to develop a DVE, which is capable of supporting four or more separate and remote generic devices. Local (client) interfaces are created for the SH-60B MRT3 and a commercial desktop based flight simulator. A visualization of the environment state is constructed using the X3D Graphics open standard for defining and communicating real-time, interactive 3D content. Test results are added to the Savage model archives and software extensions are added to the online open-source Open-DIS codebase. Finally, the overall system is demonstrated and a performance test of the system scalability is conducted.

D. THESIS ORGANIZATION

The remainder of this thesis is organized as follows.

1. Chapter II: Background

This chapter provides a general functional description of distributed virtual environments with a focus on enabling interoperability and interactivity for distributed participants. Existing standards for distributed interactive applications are briefly examined and an argument is presented for the relevance of research, which seeks to apply MMOG architectures, and middleware to military distributed virtual environments (DVEs).

2. Chapter III: Related Work

This chapter examines previous work specifically related to alternative architectures for distributed virtual environments including MMOGs and application middleware. This chapter also presents an overview of the existing Modeling & Simulation (M&S) applications used as test cases for the implementation portion of this work.

3. Chapter IV: Methodology

Details of the implementation choices relevant to this DVE-construction work are presented here. In addition to implementation of an MMOG server, the use of heterogeneous simulation applications required the construction of several gateways. Development of an open-source gateway for a high frame-rate commercial flight simulator is also presented. Since a virtual environment is not complete without visualization, the implementation of a graphical reflection of the virtual environment state using open-source, open-standards tools is presented. Methods for measuring the performance of the system architecture are also given.

4. Chapter V: Results, Performance and Behavior

Quantitative and qualitative system performance is presented in this chapter. Software profiling data of the server under different loads is presented as well as network analysis of the communication between the server and a client simulator.

5. Chapter VI: Conclusions and Recommendations

Based upon the insight gained from this work, conclusions are presented on the feasibility of using MMOG architecture for a military distributed virtual environment. In addition to the functionality demonstrated in this work, several other useful capabilities may be gained with this hybrid architecture. This work is expected to serve as the foundation for a dedicated Massively Multiplayer Online (MMO) service hosted at the Naval Postgraduate School.

THIS PAGE INTENTIONALLY LEFT BLANK

II. BACKGROUND

A. INTRODUCTION

The DoD pioneered the use of Distributed Virtual Environments (DVEs) for the training of large numbers of heterogeneous interacting units with the development of Simulator Networking (SIMNET). The design choices made for SIMNET sill heavily influence the development of DVEs today. Whether they are complex networks of military training devices or entertainment services such as Massively Multiplayer Online Games (MMOGs), all DVEs share a common set of features. One key feature is the ability of each participant to have a shared sense of state of the environment. Due to some fundamentally difficult characteristics of heterogeneous distributed systems, assuring success is an inherently complex and demanding task. The growth of commercial MMOGs means that significant resources outside of the DoD are being invested and applied to addressing similar challenges, and the technology in the MMOG sector will continue to develop more rapidly than the corresponding technology in the defense sector. It may therefore be possible to take advantage of this development with DVE architectures that are hybrids of current standards and emerging technologies.

B. DISTRIBUTED VIRTUAL ENVIRONMENTS

The scenario is dramatic. The group of tanks swiftly moves cross-country to cut off the opposing armored column. It is late afternoon but the overcast and the dust reduce visibility to almost nighttime conditions. The tank commander in the lead vehicle peers through his night vision scope trying to make out the shapes on the distant rise ahead. A quick verbal command to the gunner and the turret slews right. There, just moving from behind an earthen revetment is definitely a tank, no three tanks. He calls out a sighting report as the gunner prepares to fire. Soon the rest of the regiment would follow. The Battle of “73 Easting” had begun.

The above vignette is directly relevant to the subject of distributed virtual environments because one of the first successful distributed virtual environments (DVE), SIMNET, was used to augment the training of the 2nd Armored Cavalry Regiment soldiers that fought in that one-sided battle on February 26, 1991. The SIMNET infrastructure was also used to recreate the battle for purposes of visualization and analysis after the fact (Figure 1). The architectural and conceptual decisions made for the implementation of SIMNET still heavily influence the development of DVEs today (Lenoir, 2003; Miller & Thorpe, 1995). A functional definition is needed for the term distributed virtual environment, identifying the primary factors, which constrain their scale and consistency.

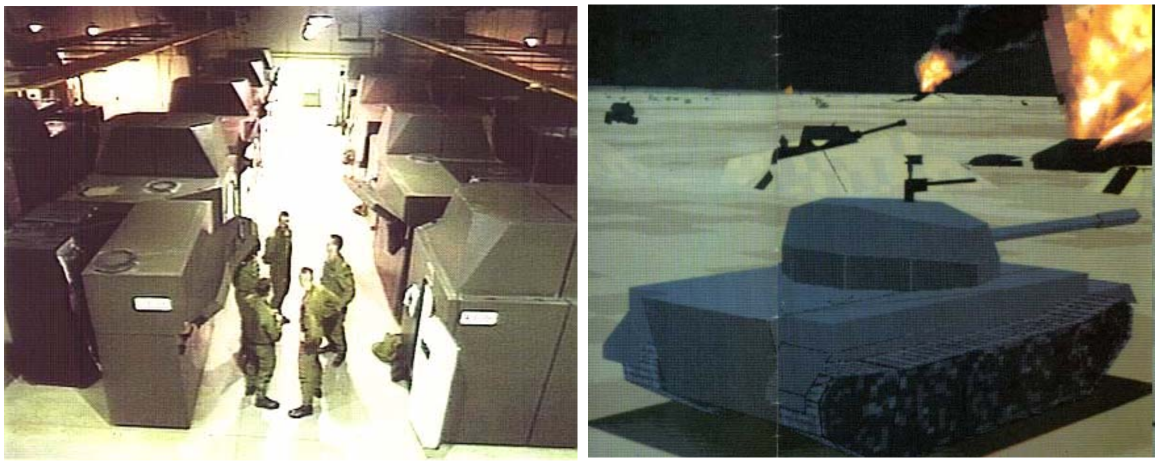


Figure 1. (left) SIMNET vehicle simulators at Fort Knox, KY. (right) SIMNET screenshot From Bruce Sterling's “War is Virtual Hell,” (From: Sterling, 1993).

Military operations by their nature involve many heterogeneous interacting units. Historically, due to system complexity and hardware costs, as well as the benefits gained by collective training, the military has been the leading pioneer in the development of distributed virtual environments (Miller & Thorpe, 1995). As hardware performance has improved to the point where even desktop computers can outperform the expensive purpose-built systems of only a few years ago, and as relatively high-bandwidth network infrastructure has become ubiquitous beyond the military domain, the commercial game industry (with some significant caveats) is now leading the development of DVE architectures (Smith, 2009; Narayanasamy et al., 2006).

Regardless of whether the system is a tool for combined arms training such as SIMNET or a Massively Multiplayer Online Game (MMOG) such as World of Warcraft®, all DVEs can be described as a system, which possesses the five following features.

- A shared sense of space: All participants have the illusion of being in some physical relation to one another whether adjacent or across a field.
- A shared sense of presence: Participants take on a persona or avatar when they enter and interact within this environment. The avatar is not necessarily a human.
- A shared sense of time: Some degree of real-time interaction and temporal consistency must occur.
- A way to communicate: Voice or message transmission among participants adds a necessary sense of realism to any simulated environment.
- A way to share: The ability to interact with other participants in the environment includes the ability to have a shared state of the environment itself (Singhal & Zyda, 1999).

In addition, four basic components are required to implement a DVE.

- Graphics engines and displays
- Communications and control devices
- Processing Systems
- Data Network (Singhal & Zyda, 1999).

As a distributed computer system, at its most abstract a DVE is a system for passing messages between computers over a network. DVE designers and developers must cope with the challenges of managing network resources, dealing with concurrency, and compensating for the variability in the transport of network traffic (Diehl, 2001). In an idealized shared virtual environment, all participants need to have completely up-to-date knowledge of the environment state and also need to interact with the environment and other participants in a natural and seamless manner. As a specific example, if participant A picks up an object, participant B (within the limits of human or game perception) needs to have knowledge of this change of environment state.

Network latency is defined as the amount of time required to transfer a data message from one point to another. In a Local Area Network (LAN), this is on the order of 10 milliseconds (ms). It is on the order of 100 ms for transcontinental transfer and 250 to 500 ms for intercontinental transfer (Pullen, 2000; Singhal & Zyda, 1999). The theoretical lower limit is based on the speed of light, approximately 10 ms per time zone. Due to network latency and processing delays, it is theoretically impossible to have a highly dynamic shared state and high consistency at the same time. This is referred to as the consistency-throughput tradeoff.

> Consistency-Throughput Tradeoff: It is impossible to allow dynamic shared state to change frequently and guarantee that all hosts simultaneously access identical versions of that state. (Singhal & Zyda, 1999)

Consistency in this context is best defined as information uniformity among the parts of a complex system. Throughput is the rate at which such shared state is updated. While graphics engines and computer processing may impose some constraint on how quickly updates to shared state may be *displayed*, it is the architecture of the DVE and the characteristics of the network itself that is primarily the cause of the consistency-throughput tradeoff (Singhal & Zyda, 1999; Pelligrino & Dovrolis, 2003; Tannenbaum & Steen, 2002).

In general, DVEs place a significant demand on network resources. Without modification, the original SIMNET architecture would require 375 Mbps of network bandwidth for 100,000 players (Macedonia, 1995). Modern MMOGs, which are relatively sparse with the amount of data that is exchanged in each message, still require approximately 2-3 Kbps of bandwidth per player (Chen et al., 2005). This is actually comparable to the bandwidth per participant required for military DVEs (Keune & Coppock). Commercial and military DVE developers face similar challenges and consequentially are forced to consider similar solutions.

Most commercial distributed virtual environments use a client-server architecture in order to maintain authority over the interactions between participants (Hall & Novak, 2008; IGDA, 2004; Bogojevic & Kazemzadeh, 2003). This includes administrative interactions such as authentication and session matchmaking, or simulation interactions such as collision detection and combat resolution. The server manages the current state of the entire world. A client sends an update to the server, which propagates it to all other clients. When all communication is via the server, the server is a potential bottleneck. In peer-to-peer architectures, every client has a partial copy of the state of the world. If something changes, the client has to send this change to all other clients. Client-server and peer-to-peer are the extremes of the spectrum of possible architectures (Figure 2). For large flexible virtual worlds, hybrids are often more appropriate (Hsu et al., 2003; Diehl, 2001).

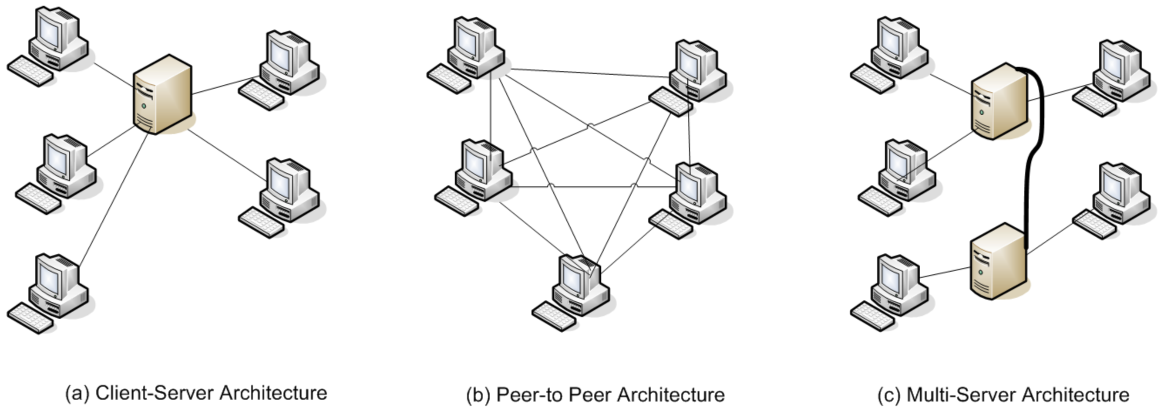

(a) Client-Server Architecture
(b) Peer-to Peer Architecture
(c) Multi-Server Architecture

Figure 2. Topologies of Distributed Virtual Environments. In practice, hybrid topologies are often used for large DVEs in the military domain. (From: Hsu et al., 2003)

The basic functional characteristics of a distributed virtual environment have been examined. One of the primary components, the network infrastructure, is generally beyond the direct control of the developer. The consequence is a fundamental limitation on a DVEs capability as described by the consistency-throughput tradeoff. Generic architectures for managing consistency and throughput usually have common architectural designs.

C. LEVERAGING THE DEVELOPMENT OF MASSIVE MULTIPLAYER ONLINE GAMES (MMOGS) FOR DISTRIBUTED SIMULATION

A massively multiplayer online game (MMOG) is a distributed virtual environment (DVE) in which a large number of people can participate simultaneously over a network, interact with each other (and usually interact with the game world), join in or leave at any time and expect everything they have produced to persist while they are offline (Hall & Novak, 2008). Applications may vary from simple text-based Multi-User Dungeons (MUDs) to visually rich role-playing games. Both extremes meet the functional requirements of a DVE that were presented above. While there has always been interest in the cognitive and social dynamics of participant interaction within shared virtual environments, recently there has been an explosion in academic and commercial interest regarding the development of both the technology to enable these environments and the content to make them viable (Bonk & Dennen, 2005). The rapid growth in the number of subscribers has consequences that extend beyond the aesthetic value or social implications of interacting with large numbers of other participants. Commercial MMOG development is being driven fundamentally because of the fact that when they are successful, they can be much more profitable than single-player computer games (Hall & Novak, 2008). As the number of subscribers grows (Figure 3), there is a corresponding push to develop the technology required to manage the large numbers of participants within individual environments, corresponding efforts are also needed to manage the significant and growing impact that the message traffic of these game has on the internet infrastructure. Developers of DVEs for military applications must both carefully watch both trends and actively participate in this process of innovation.

For a premier single-player computer game to be profitable, where development costs range from $15 to $20 million, 500,000 unit sales is the minimum needed. Some have argued that the computer game market can no longer support development of more than a few such titles (Deering, 2009). Even though first-class MMOGs may cost more to develop, often they can become profitable at only 250,000 unit sales. MMOGs can also lose much more money because of the infrastructure cost of supporting the game (Hall & Novak, 2008). Per-subscriber revenue for commercial MMOGs averages between $9.95 a month and $12.95 a month (Tay, 2005; IGDA, 2004). A naïve analysis shows that an MMOG that maintains an average of 250,000 subscribers over five years will generate on the order of $150 million in revenue. Extremely successful MMOGs like World of Warcraft® might generate this amount of revenue in a few months (Lent, 2008). The cost of operating and updating this ongoing service must also be factored into the budget equation. Arguably, this post-sale revenue potential is the key economic factor, which is driving the development of MMOG technology.

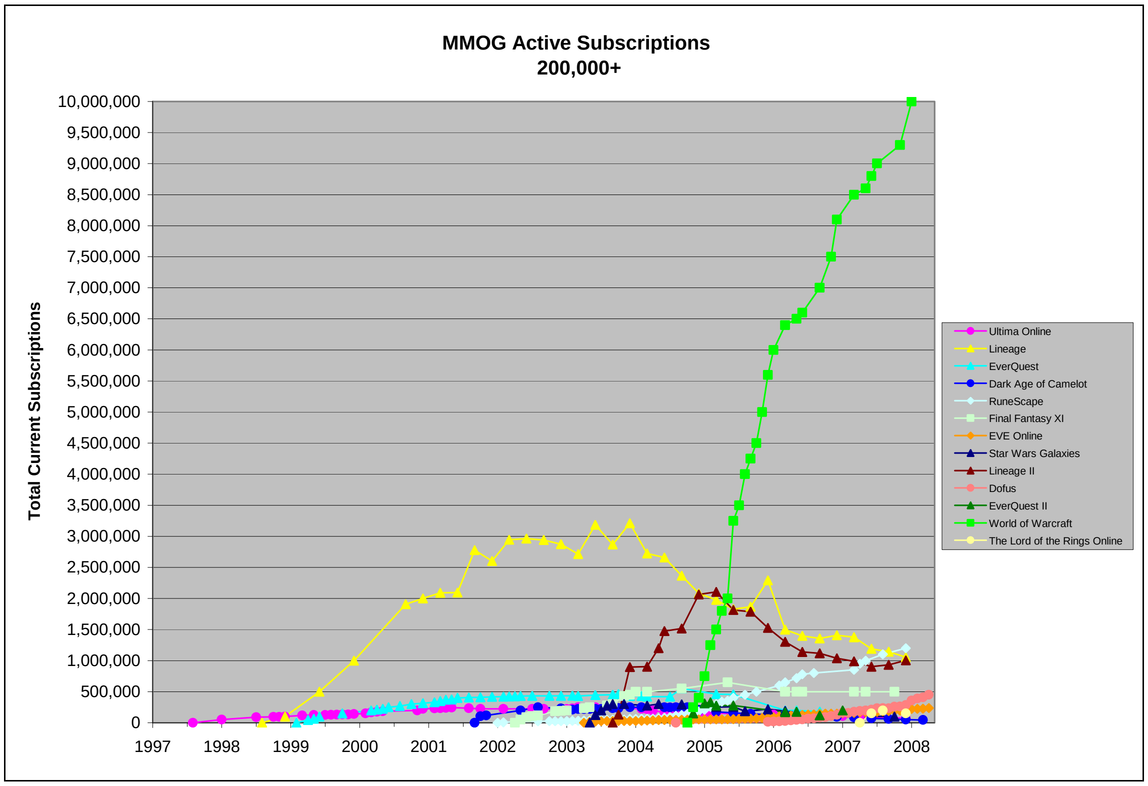


Figure 3. MMOG Active Subscriptions: (http://www.mmogchart.com). This shows the growth of MMOGs and the dominance of the top six. (From: Woodcock, 2008)

As the number of subscribers grows rapidly, it follows that the number of simultaneous users will grow as well. In general, distributed virtual environments place great demands on network resources, and such demands are expected to increase (Claypool, 2005). For example, Gamania, a Taiwanese company that operates the game Lineage, owns more than 4,000 Mbps of dedicated links for game traffic (Chen et al., 2005). This imposes a cost on the providers of MMOGs and has an impact on other users of the network. This is a real constraint on the growth potential of these environments. As has been discussed, the Internet, while generally extremely successful at scaling up to a global sized system, was not designed with distributed virtual environments in mind. As these games grow, they are likely to have an influence on the way that network infrastructure develops.

The development software required to implement such games has followed a traditional arc, from closed and proprietary software tied to a single game, to commercial toolkits decoupled from a specific game, to multiple, disparate open-source implementations. As the segment further develops, a nascent standardization and interoperability effort is beginning to coalesce in the Internet Engineering Task Force (IETF) space (MMOX Wiki), primarily around architecture closely related to Linden Labs' Second Life. The maturation of the commercial MMOG market creates an opportunity for the DoD. If a viable, standards-based community develops around MMOGs, the DoD may be able to piggyback off this effort and exploit the economies of scale, commercial tools, and base of experienced engineers produced by the commercial game industry (Bonk & Dennen, 2005). However, this opportunity must be balanced against the requirements of a large installed base of legacy M&S software.

D. ENABLING INTEROPERABILITY IN DISTRIBUTED VIRTUAL ENVIRONMENTS

Interoperability is the ability to connect simulation clients together such that those clients can operate on a perceived shared virtual environment (Zyda, 2005). A useful distributed virtual environment needs to enable heterogeneous hardware and software applications to participate in the environment. This is especially critical for military DVEs since there has been a great deal of investment made in hardware and software without consideration to DVE interoperability. Subsequently, much effort has been expended to enable interoperability in order produce a common synthetic environment (Page, 2002). This section examines both the syntactic and semantic interoperability prerequisites for the use of heterogeneous hardware and software in a DVE. This is a fundamentally difficult problem. Interoperability can also be described as the meaningful exchange of information between participants in a DVE. The Distributed Interactive Simulation (DIS, IEEE 1278) and High Level Architecture (HLA, IEEE 1516) are discussed and the ability of each standard to enable interoperability is briefly examined. The architecture of MMOGs is reviewed in general and posited as an alternative architecture for military DVEs. Finally, the necessary quality of this information exchange is discussed with specific attention to update rate and latency for shared state.

1. Motivation for Interoperability

Interoperability is a fundamental requirement of military virtual environments because military operations are highly complex and heterogeneous. Nevertheless military simulations tend to be developed for specific purposes. In addition, the idea of cost savings through reusability and composability has driven the desire for interoperability among DoD virtual environment applications (Tolk & Muguira, 2003; Davis & Anderson, 2003). Further significant benefits can occur if tactical systems become capable of interoperations with simulations systems (Brutzman et al., 2002).

DoD virtual environment applications range from relatively simple PC-based simulations to multi-million dollar training devices. Although often a great deal of effort and expertise has gone into their development, generally speaking each is developed for a specific purpose without considering for the need to interact in a meaningful manner with one another. The fact that military operations are inherently highly complex and heterogeneous means that, in order to use these virtual environments in a way that reflects real world military operations, each must be composed coherently into a distributed virtual environment (Miller & Thorpe, 1995).

The complexity of developing monolithic virtual environment systems has often led to failures in system development. A possible alternative to monolithic systems is composability, which is the selection and assembly of previously existing components to satisfy some user requirement. Interoperability is a necessary precondition for composability (Page et al., 2002). If two systems cannot exchange information in a meaningful manner then they certainly cannot interact to satisfy overall system requirements (Davis & Anderson, 2003). Therefore, composability reduces the risk of developing large scale virtual environment systems and leverages the huge investment in previously existing systems. Such composability becomes far more powerful if the functional improvements are focused on data-stream interoperability rather than codebase re-engineering.

2. Theory and Practice of Interoperability

Interoperability is defined in general as:

> The ability of systems, units or forces to provide services to and accept services from other systems, units or forces, and to use the services so exchanged to enable them to operate effectively together. (Joint Publication 1-02)

A technological definition of interoperability may be as simple as the ability to exchange data or services at run-time. For distributed virtual environments this definition must be extended because individual simulations whether they are game-based trainers or full-motion flight simulators must act on this data in order for a shared virtual environment to exist. While interconnection as a prerequisite for interoperability, there is little point to interconnected communication systems if the meaning of the exchanged information is different in each system.

A great deal of rhetoric has been published regarding to the notion of plug and play interoperability of different simulations. The actual experience of composing large numbers of heterogeneous simulations together has forced many users and developers of distributed virtual environments to re-examine the motivation and theoretical foundations for arbitrary simulator interoperability (Ceranowicz et al., 2002; Davis & Anderson, 2003). Recent work has generally concluded that it is impossible to apply a fixed standard of interoperability across all domains of modeling and simulation, and that a level-of-interoperability metric might better be used instead (Tolk & Muguira, 2003). A fundamental definition of a model is representation of an element of the real world for a purpose. A simulation is the corresponding execution and response over time. By the nature of model construction, there are usually explicit and (more importantly) implicit assumptions made that reveal gaps when the model is compared to the real world. In attempting to enable these models to interact with real world systems or other models, there is a tendency towards complexity. As model complexity increases, it soon reaches a point where the risk of failure rises much more quickly (Chwif et al., 2000).

Distributed virtual environments (DVEs) are directly analogous to command and control (C2) systems and distributed databases because they are all essentially means of sharing a common state. The notion of information-exchange requirements (IERs) is one of the central concepts in the development of these types of systems (Tolk & Turnitsa, 2007). For distributed virtual environments, the IERs can be summarized as follows.

- Syntactic: data type, message length, protocol
- Contextual/Conceptual: modeling assumptions, purpose of the information
- Semantic: units and meaning of data, conventions for internal algorithms

Syntactic interoperability is primarily an engineering problem and is generally addressed with well-defined and broadly accepted standards. Understanding of conceptual interoperability is perhaps incomplete but there is usually sufficient conceptual understanding to enable correct syntactic mappings between different data formats. Semantic interoperability is achieved by having a common method of identifying data, such as a common data model (Davis & Anderson, 2003; Tolk & Muguira, 2003). Arguably DIS and other network protocols completely address syntactic interoperability. To some extent both DIS and HLA address semantic interoperability as it is defined above. One important element of semantic interoperability is the need for a common way of specifying location as some models may use geocentric Cartesian (x, y, z) and others spherical (latitude, longitude, and altitude) coordinate systems. Middleware, distributed algorithms, and data-type coercion are generally used to solve the problem of semantic interoperability. Middleware is a software layer that masks the heterogeneity of the underlying networks, hardware, operating systems and programming languages (Britton et al., 2004; Tannenbaum et al., 2002). Another key element of conceptual interoperability is the nature of each individual model’s assumptions (Garlan et al., 1995). Both DIS and HLA presume that all participants are trusted agents, in that they do not deliberately “cheat” or misrepresent DVE state. However if one system considers clouds in visibility determination and another does not, then an unintended “cheat” has just been introduced. Simplistic solutions to these challenges are likely to fail.

The Extensible Modeling and Simulation Framework (XMSF) and the Levels of Conceptual Interoperability Model (LICM) are efforts directed towards addressing both semantic interoperability and conceptual interoperability (Brutzman et al., 2002; Tolk & Muguira, 2003). Further discussion on conceptual interoperability is beyond the scope of this thesis, except to show that alternative frameworks for distributed virtual environments supporting military applications are important topics for continued consideration. The application of standards and middleware for enabling interoperability is relevant to this thesis. In the end, what the “cooperative execution framework” of two simulations might be called is unimportant when compared to whether or not useful training can be provided by the connected systems (Page et al., 2004).

3. Distributed Interactive Simulation (DIS): A Protocol Approach

The Distributed Interactive Simulation (DIS) protocol, IEEE 1278.1a 1998, is an open-standard protocol that defines message format and content to enable the connection of simulations. It was developed specifically for real-time applications and this is reflected by the specifications of the standard. DIS benefits from being relatively straightforward conceptually and structurally, but it has some limitations, which are imposed by its real-time nature.

DIS was closely patterned after the distributed interaction protocol for SIMNET that was developed in the 1980s. SIMNET itself was developed to enable distributed interaction for large numbers of combatant entities. The original concept was eventually constrained to ground vehicles because the message-exchange requirements were more manageable (Miller & Thorpe, 1995). The success of SIMNET led to the funding of an effort to develop a standard for real-time platform-level wargaming across distributed hosts beyond SIMNET. A series of workshops hosted at the University of Central Florida led to the initial adoption of the DIS standard, which then became a formal IEEE standard in 1993. The mission of DIS is to create synthetic virtual representations of warfare environments by systematically connecting separate subcomponents of simulations, which reside at distributed multiple locations (SISO, 2007).

DIS messages are called Protocol Data Units (PDUs) and they are the means of information exchange between simulations. There are 67 standard PDUs in the current version of the standard (IEEE 1278). Non-standard PDUs that are application specific are sometimes used as well. There are four types of PDUs, which generally define the primary interactions between participants in a DIS-based DVE. These are the Entity State, Fire (as in weapon fire), Collision, and Detonation PDUs (IEEE 1278).

The Entity State PDU (ESPDU) is a fairly complete representation of the dynamic physical state of the entity (Table 1). It is an inherently platform-level representation of position and orientation based on rigid-body physics, capturing state for kinematic (velocity-based) or dynamic (acceleration-based) motion. The ESPDU contains elements such as location, linear velocity and others. At a minimum, an ESPDU will be 576 bits (72 bytes) in length. A typical ESPDU is 144 bytes long. ESPDUs generally make up more than 70% of the message traffic in DIS based DVEs (SISO, 2007).

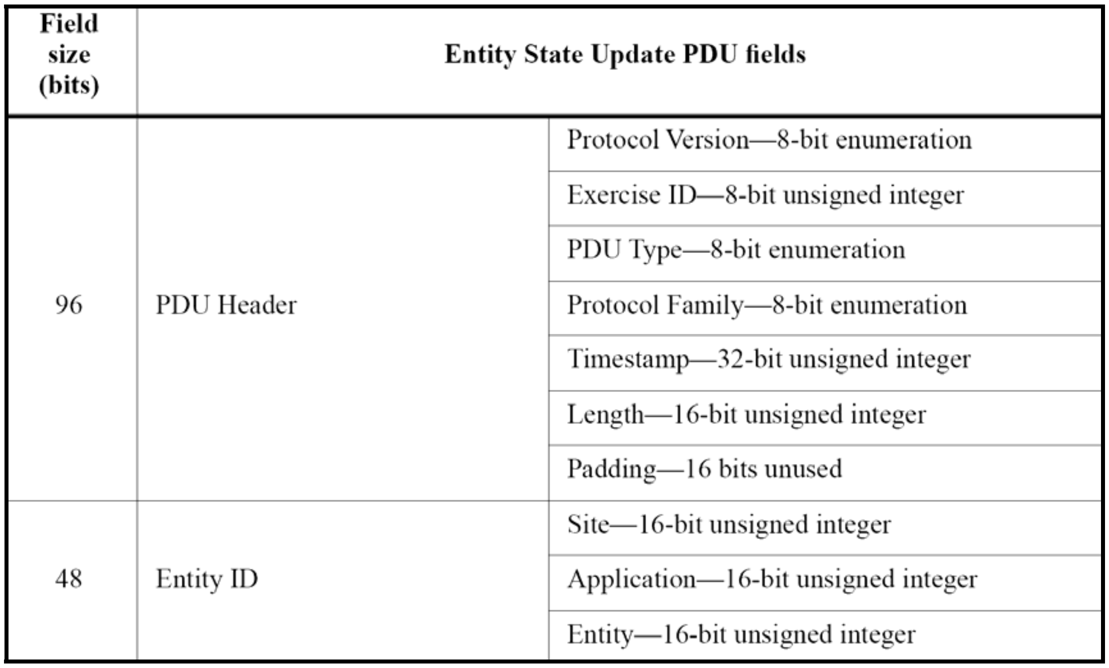
DIS requires several key assumptions in order to work that are part of the formal standard.

- Participating simulation nodes in a DIS-based DVE are autonomous and authoritative with respect to their own state, and transmit the ground-truth of that state to other nodes.
- Each simulation node is responsible for periodically issuing the PDUs that reflect that node's state.
- There is no central computer with an authoritative record of the state of that node. Each simulation node determines what is true for that node including the effect of actions by other nodes (IEEE 1278).

A specific example considering two simulation nodes (A and B) and the handling of weapons fire is descriptive. If simulation node A fires a weapon at node B, node A sends a Fire PDU, via broadcast or multicast transport. The simulation modeling the munitions fired then sends a Detonation PDU, which includes the impact location if applicable. This is typically the firing node but the DIS standard does not explicitly require this to be the case. Upon receipt of a Detonation PDU, simulation node B determines if any damage occurred as result of the detonation. It is apparent that DIS relies inherently upon completely trustworthy clients. Another key assumption in this methodology is the use of dead reckoning for updating physical location and orientation between updates, in order to reduce the number of messages that need to be transmitted and to provide a shared consistent state at all nodes.

While DIS has been relatively successful, it has some well-known limitations that are a result of the fact that it was derived from the SIMNET protocol. SIMNET was specifically designed for interaction involving dozens to hundreds of distributed real-time platform level simulations (Miller & Thorpe, 1995). One consequence of the autonomous node architecture is that many “heartbeat” state-update messages must be sent, even when there are in fact no changes to the state of the entities at that node. Even when some aspect of state does change, much of the information contained within the PDUs themselves may be the same. The result of this is that while DoD goals foresaw networked simulations involving 100,000 or more participants, in practice the highest numbers of simultaneous participants using DIS only is on the order of two to three thousand such as was seen during the Synthetic Theater of War (STOW) series of experiments in the mid 1990s (OTA, 1995; Keune & Coppock, 1995).

Table 1. The first 144 bits (18 bytes) of an ESPDU (From: IEEE 1278)



Field size (bits)	Entity State Update PDU fields	
96	PDU Header	Protocol Version—8-bit enumeration
		Exercise ID—8-bit unsigned integer
		PDU Type—8-bit enumeration
		Protocol Family—8-bit enumeration
		Timestamp—32-bit unsigned integer
		Length—16-bit unsigned integer
		Padding—16 bits unused
48	Entity ID	Site—16-bit unsigned integer
		Application—16-bit unsigned integer
		Entity—16-bit unsigned integer

Significant research has been conducted to expand the scalability of DIS-based DVEs, including an area of interest management (AOIM) schemes, which seek to only transmit state update messages to those nodes that have a need for the information, and hybrid architectures where related messages are aggregated and then sent collectively to disaggregation nodes (Macedonia, 1995). In this regard, DIS does not meet the requirement of a scalable system because the architecture must be changed in order to increase the size of the system. For small-to-medium sized platform-level real-time distributed virtual environments, DIS is completely suitable. Despite the fact that its conceptual framework is based upon a twenty-year-old protocol, it is still in widespread use in military DVEs (Strassburger et al., 2008; Steel, 2000).

4. The High Level Architecture (HLA) IEEE 1516

The High Level Architecture (HLA) IEEE 1516 is a software architecture that provides the ability to link different kinds of simulations, simulators, models and other tools. Although it is generally associated with distributed simulation, the primary motivation for its development was to support “composable” simulation (Dahmann, 1999; IEEE 1516, 2000). Although HLA is potentially a powerful architecture, the process by which it was developed and the requirements it must meet created a structure that has a steep learning curve for users. Hopefully, an emergent need for complex reusable simulations beyond defense applications might someday reduce this learning curve.

Traditional simulation models often lack two desirable properties: reusability and interoperability (Kuhl et al., 1999). In this context, reusability means that old simulation models can be reused in different simulation scenarios and applications. Interoperability means that the reusable component simulations can be combined with other models and simulations without the need for recoding. Several HLA design goals were established in response to these common needs. They may be divided into two sets.

- The need to save existing simulations
    - To maximize the reusability of the existing simulations models
    - To make it possible for individual simulation models to be combined in order to model more complex systems
    - To allow the individual simulation models to interact in a manner that supports distributed simulation technologies
- The need to allow for future capabilities.
    - To provide a larger capability for modeling command and control (C2) structures
    - To provide a flexible framework that will permit new technologies to be incorporated in the simulation models
    - To provide simulation models that can be immediately incorporated into developing planning and control technologies (Kuhl et al., 1999)

To encourage widespread acceptance of HLA, DOD submitted the HLA standards to two standards bodies, the Object Management Group (OMG) and the IEEE. The Interface Specification was adopted by the OMG in 1998. The IEEE adopted the HLA Standard in 2000 as the IEEE 1516 series. In November 2000, all of the services signed a Memorandum of Agreement that mandated HLA as the standard technical architecture for interoperability among DoD simulations (DoD A&T, 2000).

a. Features of HLA

The baseline definition of the HLA includes: HLA Rules, HLA Interface Specification, and HLA Object Model Template. It is important to note that HLA is not software per-se. Rather HLA is a set of rules. The rules are divided into two parts: the federation rules and the federate rules. Federations work with the Federation Object Model (FOM) to interoperate. The federate rules direct that each federate must document their public information as well as what they must import and export to other federates, in order to comply with the federation Run-Time Infrastructure (RTI). The RTI is the actual implementation software and hardware that links federates (Figure 4). The RTI manages six critical functions; Federation management, Object management, Time management, Declaration management, Ownership management, and Data Distribution management (Kuhl et al., 1999). For two HLA-compliant applications to interoperate, they must utilize the same FOM and the same RTI (Ryan & Zalcman, 2003). Some form of software commonality is also needed for message exchange. More detailed technical discussion of HLA is beyond the scope of this thesis.

The DoD-sponsored development of HLA was given impetus due to the belief that composability (reusability and interoperability) leads to better simulations at reduced cost. Based upon the historical trend towards increasingly complex and expensive “monolithic” simulations, this notion certainly seems reasonable. Typically technology goes through a lifecycle from birth to retirement. Initially, it may be completely within the research and development (R&D) domain, and then through the process of transfer and diffusion a new technology transitions to a point where new products are developed, ultimately to a phase where it undergoes general adoption. Usually it does not reach the phase of general adoption until tools and methods for its implementation are relatively stable and mature. This maturation process takes time and significant effort (Fowler et al., 1993). Due to the apparent urgency, initially DoD tried to supply all of the elements of the software economy that normally accompany the development of a new technology. This includes the software producers, consumers, publicity, training, support, testing, and perception of economic benefit (Kuhl et al., 1999).

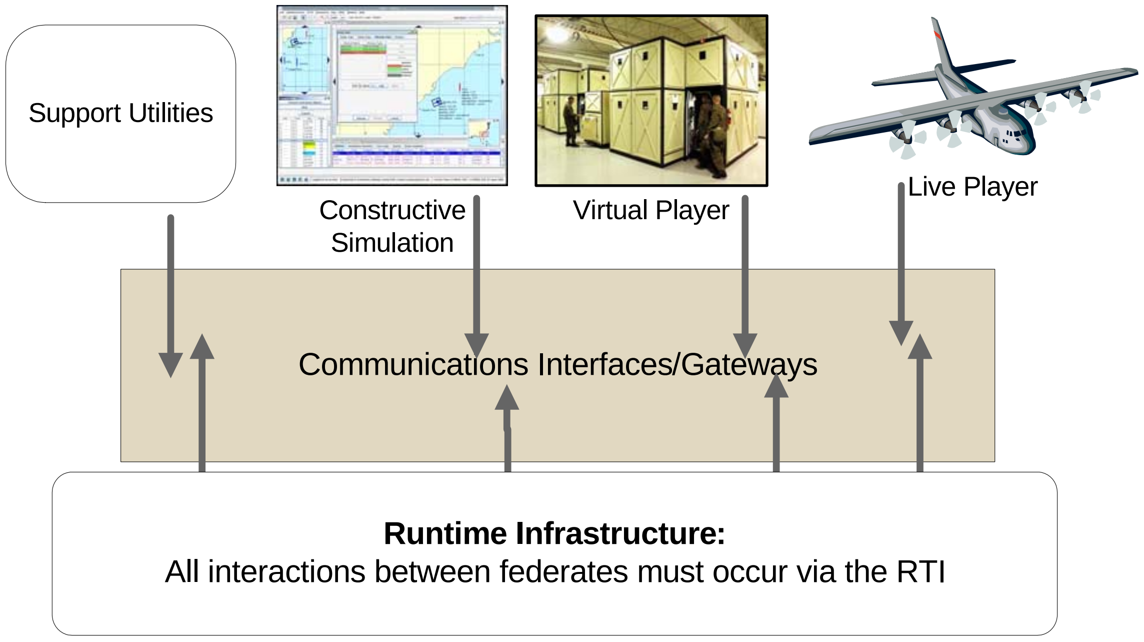

Support Utilities
Constructive Simulation
Virtual Player
Live Player
Communications Interfaces/Gateways
Runtime Infrastructure:
All interactions between federates must occur via the RTI

Figure 4. Conceptual Overview of HLA.

Generally, a Federation Object Model is described as what the various federates publish and subscribe to. This is a deceptively simple description. It follows that in order to build a federation object model, the developer must specify all of the publish-and-subscribe characteristics for every federate. Except for very simple federations this must be carefully defined beforehand. This requires a software engineering approach and a capacity for high-level thinking, and discourages rapid changes at run-time. Anecdotally, for large-scale simulations, federation development time is measured in many weeks or months (Ceranowicz et al., 2002; Kim et al., 2005).

b. Limitations of HLA

Non-military application of HLA has primarily been limited to the research community with a few exceptions. The general sentiment is that HLA offers too much irrelevant capability for industrial or commercial applications. The perception of commercial simulation developers is that there is no economic benefit to themselves from reusability and interoperability. These factors, coupled with the steep learning curve associated with competence (let alone expertise) in HLA, have made its non defense-sector reception lukewarm at best (Boer et al., 2003).

HLA was designed to support composable military simulations. HLA complexity is perhaps “as simple as possible but no simpler” (Kuhl et al., 1999) but nevertheless faces barriers to widespread adoption (Strassburger et al., 2008). A recurring theme of discussion is the desire for the subsetting of major HLA functionality to enable widespread adoption. Other architectures are capable of providing analogous if reduced functionality sets with much less complexity.

A fundamental limitation is that the HLA specification does not require HLA RTI implementations to interoperate with each other and specifies no networked data formats for interoperability between different HLA implementations (IEEE 1516, 2000). This has the result of leaving otherwise HLA-compliant simulations unable to connect with each other. Sponsors of HLA-compliant simulations thus potentially become “locked in” by commercial licenses on specific RTIs.

5. Generic Massively Multiplayer Online Game (MMOG) Architectural Description

Massively Multiplayer Online Games (MMOGs) are a subset of distributed virtual environments (DVEs). While MMOGs are obviously games first and foremost, they face information exchange requirements similar to DVEs. Strip away the game facade (Figure 5) and they are more like military DVEs than not. Some general principles of distributed virtual environments can be seen in the following examination of the components and connections of a MMOG system. This examination is conducted within the context of using MMOG architecture to enable interconnection and interoperability of militarily useful virtual environments and simulations.

Figure 5. Screenshots of two popular MMOGs. (left) Lineage® (NCsoft), (right) Navy Field® (SD Enternet).

a. Qualitative Definition of the Term Massive Multiplayer Online Game

A massively multiplayer online game is one in which a large number of people can interact simultaneously over a network. They are able to join and leave at any time and they can expect that their own game state will not change while they are offline. The game environment is "persistent." The term "large number" and the length of time, which makes a game "persistent," are application dependent and not rigorously defined. Typically, most commercial MMOGs will last several years, and some have been operating continuously for almost a decade (Hall & Novak, 2008). Contrast this with the typical military distributed virtual environment whose persistence is usually measured in hours or days at most. With respect to the "massive" qualifier, any hard number selected is often completely arbitrary. Multiuser virtual environments of any size have the same fundamental functional requirements. It is also important to note that while many MMOGs may have hundreds of thousands of subscribers, and some have seen more than 50,000 users online simultaneously, none allow more than a few thousand participants to <u>simultaneously</u> interact with one another. This is comparable to military DVEs such as the Synthetic Theater of War (STOW) experiments (Feng et al., 2007; IGDA, 2004; Keune & Coppock, 1995).

MMOG developers are not free from the consequences of the consistency-throughput tradeoff. Many of the load-management schemes employed in the development of large experimental military distributed virtual environments such as area-of-interest management (AOIM) and hybrid architectures are also employed by commercial MMOGs (Bogojevic & Kazemzadeh, 2003; Macedonia, 1995; Lecky Thompson, 2009). While other military synthetic exercises such as Millennium Challenge 2002 have included as many at 30,000 participants, they were not interacting directly with one another. Separate federations of participants were used to feed command and control systems primarily for visualization (Ceranowicz et al., 2002). The qualifying characteristics of an MMOG are therefore persistent state and inherent scalability (IGDA, 2004). Scalability explicitly means that the system can handle larger and larger numbers of participants gracefully or without more-than-minor changes to the system architecture (Diehl, 2001; Singhal & Zyda, 1999).

b. MMOGs Are a Game Service

MMOGs are games. This seemingly obvious statement serves to emphasize that their design and system architecture are driven by this reality. Briefly, a game can be described as a rule-based system in which the player(s) must overcome some obstacle or challenge to reach the predetermined end-state of the game. Game actions are typically constrained by well-defined rules and a story or narrative that sets the abstract context for the challenges, the rules, and the end state (Narayanasamy, 2006; Lindley, 2003). Commercial games are designed largely to maximize sales, repeated usage, and profits. MMOGs primarily differ from stand-alone games in that they are inherently more of a service than a product. Figure 6 depicts a generalized MMOG service architecture. It follows that a major part of any MMOG system is customer management. While a complete implementation must include a significant customer-management component, only the game-services component is directly relevant to this thesis. Game services allow players to interact with other participants and enable player actions to change the game state. Since it is difficult to enforce a consistent game experience among different system architectures, almost all commercial MMOGs today are client-server systems (Hall & Novak, 2008; IGDA, 2004).

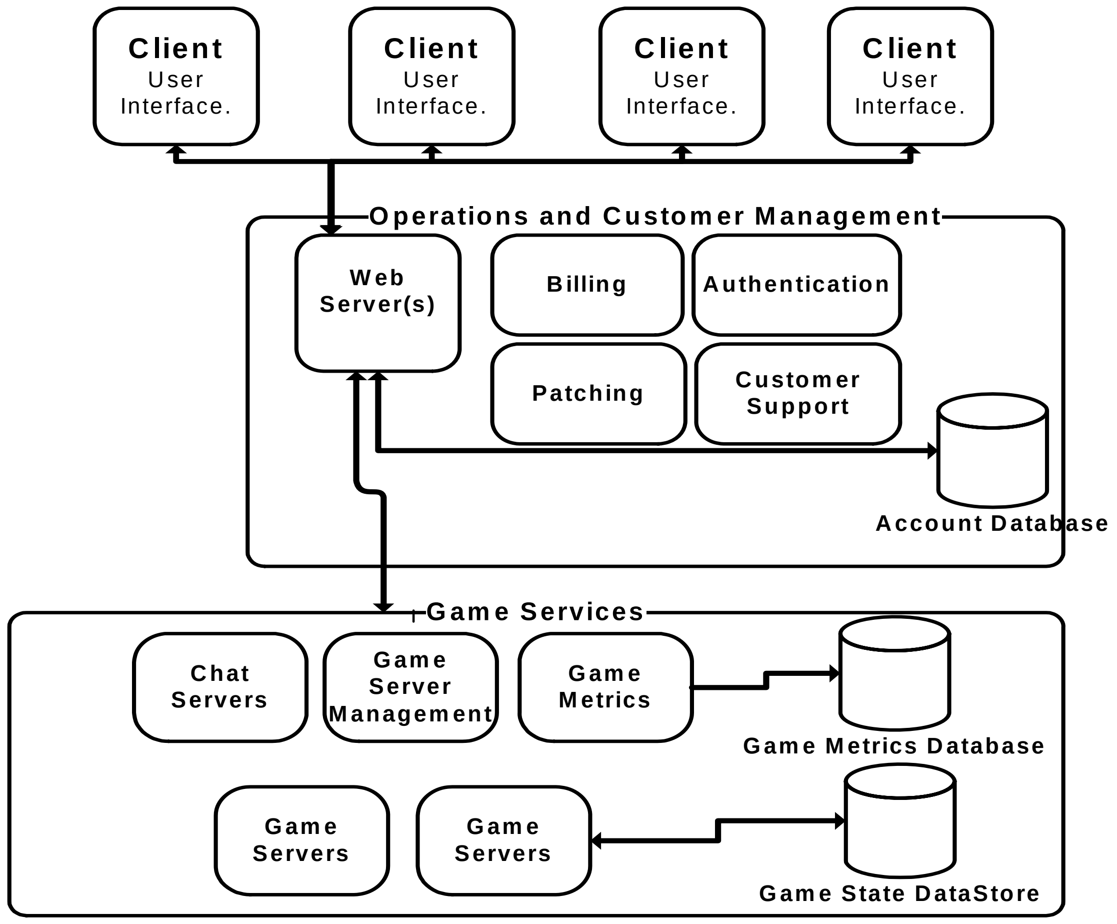


Figure 6. Abstract architecture for massively multiplayer online game (From: Hall & Novak, 2008).

Game services are described schematically in Figure 7. They are summarized as follows.

- In addition to managing the user interface, the client process the users input in the form of state update messages and transmits them to the server.
- The server then calculates a new game state by applying the received user actions and the game logic to the current game state. As a result of this calculation, the state of several dynamic entities has changed.
- Finally, the new game state is transferred back to the clients (Bogojevic & Kazemzadeh, 2003).

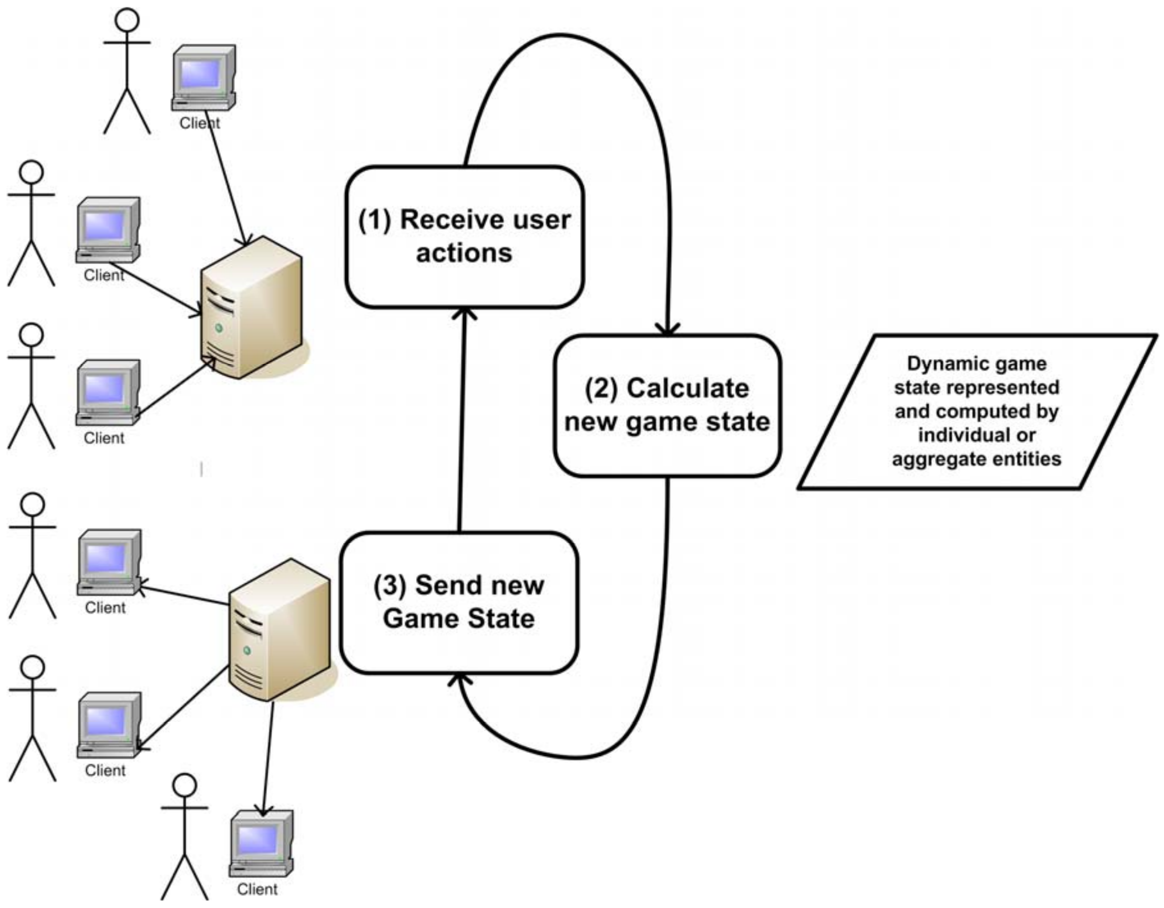

Client
Client
Client
Client
Client
Client
(1) Receive user actions
(2) Calculate new game state
(3) Send new Game State
Dynamic game state represented and computed by individual or aggregate entities

Figure 7. The fundamental game services of a client-server based distributed virtual environment (From: Glinka et al., 2007).

c. *MMOGs and Scalability*

Commercial MMOGs almost all have client-server architecture for reasons outlined earlier. However, it is obvious that a single server cannot handle an infinite number of client connections; CPU cycles on the server will eventually run out, or the server will be unable to handle a large number of socket connections or bandwidth availability becomes saturated. These problems lead to a scale-out solution, in which a cluster of computers on the server side each handles separate client connections and game state. This can be done in a variety of ways. One of the most popular is “sharding” in which parallel instances of the game state are each handled by a single server node. One of the disadvantages of sharding is that participants on separate shards cannot interact with one another (Ye & Cheng, 2006; Emilsson et al., 2009). Zoning is another load distribution scheme in which participants are assigned to server nodes based upon the participant's virtual location in the game environment. The overall game-play area is divided up into multiple geographic regions, and players are assigned to a server dedicated to each virtual region. However, this approach can have its own problems, including low server utilization. A server must be associated with a region even if few people are there at the time, which can lead to over-provisioning the data center (McGregor et al., 2009; Nae et al., 2008). Often, zoning and sharding are combined with popular game regions being duplicated over several server nodes (Glinka et al., 2007; Lu et al., 2006; Ye & Cheng, 2006).

There are a few alternatives such as the single-shard architecture used for the space-based MMO Eve Online. Eve Online applies a design philosophy that holds that the game participants are the primary content. The developers take advantage of the fact that the distance scales in the game environment create natural aggregation points at certain game locations. In this single-shard architecture, there is only one instance of the game state and any player from the global base may interact with any other player provided they are at the same game location. In the case of Eve Online, an exceptionally large cluster is used to achieve this scale (Emilsson et al., 2009).

Despite the differing motivations for their development, militarily useful distributed virtual environments and MMOGs faces similar challenges. The next section examines the similarity of the information-exchange requirements between typical MMOGs and military DVEs.

6. Comparison of MMOG Network Traffic to Military DVE Traffic

The consistency-throughput tradeoff means that developers of DVEs cannot arbitrarily decide upon the rate at which information flows between the participants. For commercial MMOG developers, who desire to handle large numbers of simultaneous players while maintaining a consistent game experience, this has a direct economic impact. A look at the network traffic patterns of various DVEs shows that with regard to network traffic, MMOGs and some large scale DVEs are analogous. While high update rates may be required for small-scale fast-reflex DVEs such as first-person shooter (FPS) type games or air combat maneuvering, it is often true that for loosely coupled interaction relatively modest update rates are acceptable (Bonk & Dennen, 2005).

Even during the development of early DVEs such as SIMNET and NPSNET, the importance of measuring network traffic was realized (Macedonia, 1995). Due to the impact that game traffic has on networks and the cost of maintaining sufficient capacity to handle large numbers of simultaneous participants, the body of research on game traffic analysis and quality of service requirements (QoS) for games has grown in parallel with the growth in the number of MMOG subscribers. This type of research can be broadly categorized into two categories. The first is the experimental determination of the effects of latency and throughput on the DVE user experience, and the second is the characterization of the network traffic of existing DVEs (Kwok, 2006).

a. Tightly and Loosely Coupled DVEs

The notion of coupling is important. This term is borrowed from systems engineering and distributed computing and refers to the degree that the behavior or processes of a node depends on knowledge about the behavior of another node. In a DVE, entities are either closely or loosely coupled. An example of tight coupling might be a group of tanks in a closely spaced formation moving at high speed. In order to maintain safe spacing and fidelity, and to avoid interfering with mutual fields of fire, each entity needs to have sufficiently accurate and timely information about the other entities in the formation (Chassot et al., 1999; Rushby, 1994). In general, tightly coupled systems do not function well when message delays exceed 100 ms. In contrast, an example of loosely coupled entities might be two tanks 20 kilometers apart. The notion of coupling is important because network QoS requirements are generally determined by whether entities are tightly or loosely coupled. Note, however, there is no hard line dividing tightly and loosely coupled systems. A qualitative distinction is typically decided by the interaction needs of the application models (Kuiper & Lemmers, 2000; Singhal & Zyda, 1999).

b. Delay and Throughput for Typical Military DVEs

The acceptable delay and throughput for a typical military DVE needs to be considered for any effective application. Unfortunately, most of the literature on acceptable QoS for military DVEs is experiential in nature. Nevertheless, experience has shown that delays of less than or equal to 100 milliseconds (ms) are acceptable for tightly coupled real-time platform-level DVEs (Chassot et al., 1999; DIS Steering Committee, 1994; Singhal & Zyda, 1999). Throughput for tightly coupled real-time platform level DVEs depends upon the information-exchange requirements for the type of platform that an entity represents. By utilizing traffic management schemes such as dead reckoning, the acceptable throughput ranges from one update per second for ground vehicles to 30 updates per second for articulated human entities (Singhal & Zyda, 1999; Macedonia, 1995).

For loosely coupled platform level military DVEs, delays of up to 300 ms are acceptable. Acceptable throughput rates again depend upon the type of entity being simulated. Unfortunately, the limited amount of research on acceptable QoS for military specific DVEs does not differentiate between throughput requirements for tightly or loosely coupled entities. Experience from the STOW experiments indicates that one to three updates per second are acceptable in this case (Keune & Coppock, 1995). Analysis that only reports average update rates should be used with caution, as this metric typically includes periods of time in which entities were stationary (Macedonia, 1995; Cheung & Loper, 1994).

The above examples are typical. An extreme case of a tightly coupled system includes remote surgery applications with haptic feedback. Minimum acceptable throughput for this type of application is on the order of 1000 updates per second (Yap et al., 2005; Choi et al., 2004). At the other extreme, computer-assisted exercises in which all interaction between entities is highly abstracted can require relatively infrequent updates from the participating entities (Ceranowicz et al., 2002).

c. Throughput and Delay in MMOGs

There are several papers describing experimental determination of acceptable delay and throughput for networked games in the literature. Not surprisingly, acceptability is broadly categorized by game genre.

d. Massively Multiplayer Online Role-Playing Games (MMORPG)

Massively multiplayer online role playing games (MMORPG) are best described as loosely coupled distributed virtual environments even though they have significant interactivity. This is because game interaction is event based. When a player clicks on a monster to attack it, an attack message is sent. The combat resolution of success or failure then occurs at the server. There is no parallel processing between the client and the server to handle the flight of an arrow for example. A typical MMORPG is Everquest II (Sony, 2009). For these types of games, investigators have reported degraded user experience at delays of approximately 200 ms (Ries et al., 2008; Chen et al., 2006). In some cases, however, delays on the order of 1000 ms have been demonstrated to have almost no effect on self-reported user experience (Fritsch et al., 2005). Within this genre, update rates range from three per second for stationary entities to seven per second for entities involved in game combat (Park et al., 2005; Chen et al., 2005).

e. Massively Multiplayer Real-Time Strategy Games (MMRTS)

Massively multiplayer real-time strategy games (MMRTS) are in relative infancy compared to MMORPGs. Typically, the player does not closely control a single avatar but exercises general direction and oversight over a large number of entities. Networked real-time strategy games (RTS) are quite mature but these are limited to a few dozen players. Warcraft III is a typical example. For these types of games, user experience often degraded once interaction delays exceeded 800 ms. In at least one case, user *effectiveness* at a set of specified in-game tasks remained unaffected with update delays of up to 2000 ms. The minimum rate for real-time strategy games is about four updates per second, with six updates per second being typical. Peaks of up to 15 updates per second are seen during combat sequences involving large numbers of entities at each node (Claypool, 2005; Bettner & Terrano, 2001). Unfortunately, the literature that covers this genre is not yet directly relevant to the challenges of MMRTS because only conventional (non-massive) real-time strategy games have been studied and reported. Developers of conventional RTS games are able to use schemes such as lock-step simulation processing. Roughly, instead of passing the status of each unit in the game, the exact same simulation is run on each machine with only commands being distributed to the other participants (Bettner & Terrano, 2001). This is only practicable for session-based games where all participants begin with the exact same state.

f. Platform-level Simulation Games

Platform-level simulation games include both vehicle simulators and so-called first-person shooters (FPS). For vehicle-based simulations such as driving simulation games, user experience and performance are relatively unaffected by delays of up to 100 ms. Beyond this point, however, performance degrades rapidly. Lap times around a simulated track in one study were 100% worse when user-interaction delay went from 150 ms to 250 ms. For car-racing games, 12 updates per second (~ 80 ms delay) is a typical throughput rate. Less-dynamic platform-level simulations do not require as high an update rate (Pantel & Wolf, 2002; Yasui et al., 2005). For FPS games, in at least one case, no degradation to player effectiveness at game tasks is experienced with delays up to 100 ms. Player effectiveness degrades rapidly for precise tasks such as precision shooting above 100 ms latency. Networked FPS games have update rates of from 20 to 40 updates per second (Beigbeder et al., 2004). These demanding throughput requirements make implementation of a scalable FPS problematic. Planetside®, a science fiction combat-oriented MMOG in which hundreds of players can participate in a single fight, is a notable example of a FPS MMOG (Sony, 2009).

g. The Effect of Architecture on Bandwidth Requirements

The state-update behavior per node for military DVEs and MMOGs has long been considered. Bandwidth requirements are dependent upon per-node behavior, the information-exchange requirements of the participants in the DVE, and the network architecture (Pelligrino & Dovrolis, 2003). Since game developers control the implementation of all nodes, the information exchange requirements for games are significantly smaller than that for militarily useful DVEs. Game nodes usually know exactly the type and behavior of all possible entities at every other game node. For online games, small packet size is used in order to reduce bandwidth and latency (Chen et al., 2005; Feng et al., 2005). A typical update message from the client to the server in a MMORPG is 30-60 bytes long following the protocol header. Compare this small size to a typical DIS ESPDU at 144 bytes. At six updates per second, the bandwidth impact is on the order of two Kbps/sec. Since servers must send partial or complete updates of complete relevant game state to clients in MMOGs, server-to-client messages vary widely in size. Some comparisons between HLA and DIS have shown that their bandwidth requirements are similar (USAF ASC, 1998).

These considerations naturally brings up the subject of DVE architecture. As discussed, MMOGs almost exclusively use client-server architecture in order to control the user experience. This means that the average bandwidth requirement at the server is a multiple of the number of clients connected to that server. This does not consider peaks in traffic for which sufficient capacity must be available. Even a dedicated T1 line (1.544 Mbps) to the server will only accommodate on the order of 600 simultaneous clients (Macedonia, 1995). It is assumed that some sort of interest management occurs at the server to prevent overloading the client's network capability. For a strict peer-to-peer architecture using DIS in which there is no server, every client broadcasts its state update messages in the form of ESPDUs. Six hundred such clients might theoretically require on the order of 3 Mbps of dedicated bandwidth *at every peer*. Clearly, this is unworkable in a wide-area network (WAN) environment. In fact, entity update rates vary significantly and in practical large-scale DVEs, hybrid architectures are used instead (Diehl, 2001; Singhal & Zyda, 1999).

h. The Question of Acceptability

Many MMOGs are able to maintain high levels of interactivity at what might normally be considered unacceptable QoS. Acceptability is determined by the objective of the specific DVE being considered. With current networking technology it is difficult to achieve tightly coupled entity-level simulation in real time for numerous entities in a wide-area network (WAN) environment. In fact, most large-scale widely distributed military DVEs used to date have been focused upon training objectives other than platform-level crew proficiency. Distributed simulation is primarily used as a tool for the training of multiple crews or staff as a coordinated military unit. The major interest has been focused on collective and not individual training. Even SIMNET, the progenitor of platform-level large-scale DVEs, had a design goal to make the crews and units, not the device, the center of attention in the simulation (Miller & Thorpe, 1995). While military DVEs have increased information-exchange requirements because of the need to support interoperability, this is a difference in degree and not type between militarily useful DVEs and commercial MMOG architectures.

7. Comparing the DIS/HLA Conceptual Model to MMOG Architecture

DIS and to a lesser extent HLA conform to the concept that autonomous nodes are trustworthy clients and they are authoritative over their own state. Games assume that some or many clients will cheat and for this reason enforce consistency at the server. In fact, cheating can occur without intent even by trustworthy clients because of differences in combat models. In effect, clients can cheat even though they are trustworthy, due to the nature of unexpected interaction consequences that emerge from different combat models or differences in terrain representation (Johnson et al., 2004).

Existing military DVEs and simulators cannot be arbitrarily connected using MMOG architecture without breaking some of the governing standards of both DIS and HLA. In fact a great deal of middleware has already been developed to enable more useful DVEs while maintaining consistency with existing standards. MMOG systems can be employed as middleware or as the core of a military DVE to improve their utility and ease of use. The similar nature of the use of the DIS protocol and architecture makes it particularly appealing for adaption to MMOG approaches.

E. APPLYING MMOG ARCHITECTURE TO MILITARY DISTRIBUTED VIRTUAL ENVIRONMENTS

Upon considering this survey analysis, the first basic use that can be made of an MMOG system for a military DVE is as middleware that connects and mediates between distributed entities interacting in a physically realistic virtual environment and a scalable MMOG server.

1. Use Cases for MMOG Middleware

The lowest level is to implement a MMOG system as a DIS-Packet Server. The motivation for this implementation includes the following.

- Abstracting away the networking component
- Allowing persistent server-side entities with their own behavior
- Implementing interest management schemes on the server side
- Development of a client-level DIS Gateway, which aggregates traffic and sends it over a single channel to the server that then interacts with another server (Singhal & Zyda, 1999)

The next level is to add game logic on the server to provide the following.

- Enforcement of consistent state via standardized damage resolution
- Collision checking
- Sophisticated server side entities
- "Gap" frame generation via dead-reckoning of client entities at the server (Fujinoki, 2006)
- Logging and reporting of DVE activity

2. Using an Open-Source MMOG Package to Implement Simulation Middleware

Project Darkstar, to be examined later, is a free and open-source MMOG toolkit from Sun Microsystems. For the purpose of this section, the Project Darkstar middleware is abstracted as a mechanism for managing the distribution and maintenance of the DVE state. A real-time platform level DVE using DIS is considered.

This work is specifically focused upon DIS because, for real-time platform-level simulations, DIS is completely suitable and this type of simulation encompasses the majority of military simulations. DIS traffic can also be used to reflect ongoing status changes in command and control (C2) systems. Existing heterogeneous platform-level simulations and tactical systems might thus become interconnected using MMOG architecture. This work shows that this type of architecture can be implemented without violating the rules and assumptions of DIS. Ongoing work at NPS is investigating bridging between DIS-style network protocols with military C4I data streams by building a track-data conversion hub.

F. SUMMARY

DIS, which has its genesis in SIMNET, was designed to support real-time platform level distributed virtual environment of at most a few hundred entities. Alternative architectures have made larger systems possible if unwieldy. HLA defines mechanisms beyond a platform-centric model to enable composability, but in practice, has not proven significantly more scalable or robust than DIS. Evidence of this is the fact that DIS is still commonly used for real-time platform level distributed simulation within the Defense sector and integrated within HLA-based simulations. MMOGs are growing rapidly and are receiving significant interest and attention. Although facing similar challenges with developers of MMOGs, developers of militarily useful DVEs must also be concerned with interoperability. Nevertheless, it is possible and advisable to apply commercial MMOG development to military DVE applications. Various tradeoffs and alternative approaches provide a rich basis for continued work.

III. RELATED WORK

A. INTRODUCTION

Large-scale DVEs and MMOGs are not useful or interesting simply because they might work better than systems, which allow only a few participants. Generally, other advantages dominate. Their strength lies in bringing physically distributed participants together in a shared environment with many potentially complex interactions. This has many helpful uses ranging from knowledge solicitation to distributed mission operations. There are ongoing efforts to mature new scalable hybrid architectures as well as to develop middleware that abstracts some of the complexity of MMOG implementation away from the application developer. Open standards allow existing simulations and models to be incorporated into a heterogeneous DVE.

B. OFFICE OF NAVAL RESEARCH (ONR) MASSIVE MULTIPLAYER ONLINE WARGAME LEVERAGING THE INTERNET (MMOWGLI)

1. Purpose

Massive Multiplayer Online Wargame Leveraging the Internet (MMOWGLI) is an Office of Naval Research (ONR) concept-development project that seeks to marry the concepts of immersive alternate reality games with massively multiplayer online games, demonstrating their application to a real scenario of interest to the Navy. An alternate reality game is one that uses the real world as a platform. Conceptually, they attempt to combine the elements of real life with an interactive artificial narrative (IGDA, 2006). It is expected that by massively scaling up the participant pool, the ability might be gained to explore more novel combinations and complex interactions of ideas, producing insights that are otherwise not forthcoming, or might be hard to predict using traditional methods. Continuous on-line play is intended to significantly reduce the fixed overhead cost of conducting a scenario-driven exercise (Jensen, 2008).

2. Implementation and Looking Ahead

MMOWGLI is not a technology development project. The objective of the project is to utilize existing technologies in a new way. The discovery phase began in 2008 with a detailed exploration into the use of MMOGs for other than entertainment purposes.

The purpose of MMOWGLI is enhanced distributed collaboration specifically for knowledge solicitation. ONR has developed several detailed notional scenarios that serve as use cases for the MMOWGLI project. This project is a credible use of MMOG technology because it focuses upon social interactions and not tightly coupled real-time interaction. While the focus is on role-playing, a rich shared environment might even allow the participation of platform level entities and their crews as stand-ins for computer-controlled units. Many interesting outcomes are possible.

C. USAF DISTRIBUTED MISSION OPERATIONS ENVIRONMENT

1. Program Purpose

The Distributed Mission Operations (DMO) Environment is a persistent training simulation network that provides an on-demand virtual/constructive environment for mission training of air crew and C4I operators. It is specifically able to operate in an environment over WAN distances, through firewalls, and constrained by high latencies (USAF ASC, 1998).

2. Current Architecture

DMO uses the Distributed Missions Operations Network (DMON), which consists of 60 nodes, at over 30 locations worldwide. DMON is a Virtual Private Network run from a Network Operations Center (NOC) in Orlando, FL (Figure 8). The individual endpoint Mission Training Center (MTC) LAN receives all outbound network traffic from the simulator and forwards it to a stream manager component that determines the appropriate data stream(s) needed for each of the participants. After stream duplication, a distributor component then distributes the stream among the remote MTC sites (McVearry et al., 2008). DMO is expected to always be a world in which multiple standards will overlap and co-exist. Specifically, both HLA and DIS are used in a federation of simulation systems. There is no indication that DMO will use less DIS in the future.

One of the primary component groups of DMO is the Test and Training Enabling (TENA) middleware, repository, and logical range data archive (Noseworthy, 2008). This system is only superficially similar to HLA and requires a gateway to participate in an HLA federation.

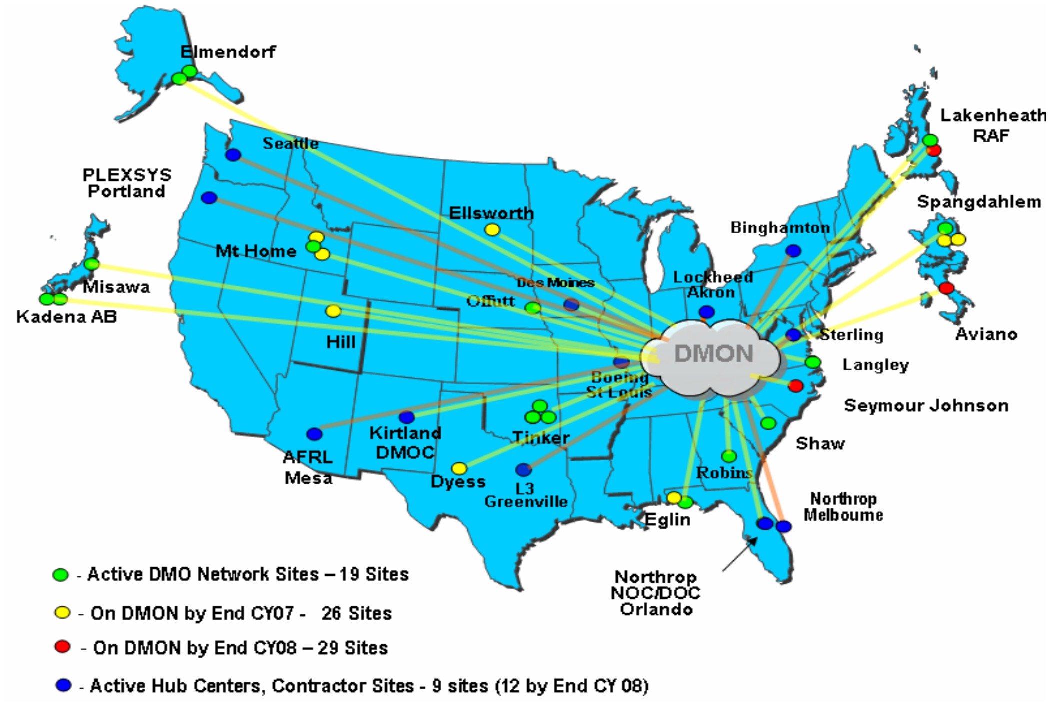


Figure 8. Distributed Mission Operations Network. (From: McVearry et al., 2008)

3. Future Possibilities

Analysis of the potential of DMO anticipates that commercial virtual environment technology will leapfrog the DoD because of the tens of thousands of developers working to advance the commercial field. It is envisioned that a hypothetical fusion of World-of-Warcraft like technology and a Second Life type system all built upon real-world databases that accurately model terrain, friendly and enemy forces has great potential to enable the scaling and extension of DMO (McVearry et al., 2008).

D. ALTERNATIVE ARCHITECTURES FOR PERSISTENT VIRTUAL ENVIRONMENTS

There are many research efforts directed towards developing alternative architectures for distributed virtual environments. These efforts can be generally categorized into two groups.

- Hybrids between client-server and peer-to-peer architectures
- Development of protocols specifically designed for distributed virtual environments.

The following are descriptions of representative examples.

1. Brigham Young University (BYU) Hybrid Game Architecture

The goal of the hybrid game architecture developed at Brigham Young University (BYU) is to reduce bandwidth requirements for MMOGs while maintaining central control. Central control is required in this architecture because it is important to control the sequence of events in the game, enforce the game rules, and update the game logic and environment. This hybrid architecture possesses the following conceptual architecture.

- Only state-changing moves are processed by the central game server
- Position changes are distributed by clients on a peer-to-peer level based upon availability and game world proximity

The guiding principle of the BYU system is that only clients that are proximate in the game environment are concerned with the most up-to-date position of one another. Moves such as attempts to retrieve objects or take some combat action are all controlled by the server. In summary, movement messages are distributed peer-to-peer to other nodes that are in the same virtual region. All other messages are distributed client-server including movement messages for nodes not in the same virtual region. Experimental results show that bandwidth requirements at the server decrease as player density increases within a game region. When player density is low within a game region, the central server is overwhelmed because very few movement messages are distributed peer-to-peer. In other words, only a small portion of the workload is offloaded to the clients.

This particular implementation of hybrid client-server and peer-to-peer architecture suggests that this type of system is suitable for DVEs where player density within regions will typically be relatively high and is not worth special implementation effort when this is not the case (Jardine, 2008).

2. The University of Pennsylvania Peer-to-Peer for Massively Multiplayer Games

Efforts at the University of Pennsylvania have produced another client-server, peer-to-peer hybrid architecture that uses a slightly different conceptual framework. Instead of having every client send non-persistent state updates (e.g., movement) to every other client that is proximate within the game region, this architecture uses a self-organizing peer-to-peer overlay based upon both the game region proximity of entities and the physical network connection between the clients, which control them (Knutsson et al., 2004).

The architecture automatically selects trusted clients to act essentially as servers for complete game-state distribution where selected clients are proximate in the network itself, not necessarily in the game environment. This differs from the system described earlier in that complete state is distributed albeit to a select number of participants. FreePastry, which is an open-source version of Pastry (Microsoft, 2009), a peer-to-peer overlay and routing network (whose detailed description is beyond the scope of this thesis) was used to implement a simple Multi-User environment.

Position updates are multicast every 150 ms to replicate the behavior of MMORPGs. Network simulation results with 4000 entities distributed across 100 to 400 regions using a single server showed that message latency is within an acceptable range for most MMORPGs. System performance suffered when player density was nonuniform from region to region (Knutson et al., 2004). While this effort is in its early phases, it suggests an alternate type of hybrid architecture that is worth considering for developers of militarily useful DVEs.

3. MMOX: Live Entity State Stream Protocol

The popular online virtual environment Second Life (Linden Labs) has received a great deal of popular attention. There are over 15 million registered accounts but little authoritative data on actual ongoing usage. Linden Labs self-reports that on average approximately 38,000 players are simultaneously connected. Second Life is interesting because it uses a slightly different information exchange framework than typical MMOGs. Analysis of Second Life network traffic shows that it is atypical for MMOGs (Kumar et al., 2008). The developers of Second Life do not describe it as a MMOG and in fact, it does not conform to all characteristics such games that are described earlier.

MMOX is an Internet Engineering Task Force (IETF) standards development project for Massively Multiplayer Online applications of all types (IETF-MMOX, 2009). The MMOX Live Entity State Stream (LESS) Protocol, which is derived from the update protocol used for Second Life, is geared toward high-rate virtual world object update. The objective of the MMOX standard development is to enable interoperability between different DVEs. It is specifically intended to enable participants to move seamlessly from one DVE to another with their persistent virtual persona or avatar. While the bandwidth requirements for LESS (approximately 100 Kbps) may limit its utility for widely distributed military specific DVEs, this effort is an important development (Rosedale et al., 2008)

E. OTHER MIDDLEWARE FOR PERSISTENT VIRTUAL ENVIRONMENTS

Quite apart from the difficulties of game development, MMOGs add the challenges of distributed-client connection handling, load balancing, shared object contention, and synchronization of data. Despite many project attempts, there has been a significant amount of attrition in this area, and much past and recent literature describes failed or stalled efforts. As a result the emerging popularity of MMOGs has led to a lot of interest in the development of MMOG-specific middleware applications. In addition to Project Darkstar described earlier, the Real-Time Framework being developed at the University of Munster in Germany is another such middleware effort. There are also some efforts to apply the DoD HLA to the domain of MMOGs. Projects that are intended to be complete MMOG solutions such as Multiverse are not within the scope of this thesis, though it is recommended that readers compare this type of software to the middleware described here (Multiverse, 2009).

1. University of Munster: Real-Time Framework

The Real-Time Framework (RTF) project is being developed at the University of Munster in Germany as part of the edutain@grid (www.edutaingrid.eu) project to support development of large multiplayer virtual environments. Its goal is to liberate the developer from low-level tasks such as multi-server communications, object migration across servers and load balancing. The RTF provides a high-level communication and computation middleware for single-server and multi-server online games (Glinka et al., 2007). The distinction of RTF is that is abstracts all of the components of MMOG development including the game engine, game logic, and game state distribution. RTF supports three parallelization schemes that are common to MMOGs; zoning, instancing, and replication.

The RTF project has not yet published implementation details beyond this level and it remains unclear whether or not the implementation or source code will be publically available. Such availability is worth monitoring because it appears to be well supported, and if successful may provide a useful platform upon which to base scalable DVEs of various types.

2. Cybernet OpenSkies MMOG Middleware/Runtime Infrastructure

While HLA is a specification, any particular HLA implementation is by definition a middleware application because it is used to connect simulation applications to the network/communications infrastructure. HLA is fairly complex yet most of its functionality is not needed for game development. Cybernet Systems, which has developed complete HLA RTIs and gateways for use in federating military simulators, developed a mini-RTI designed for MMOGs called OpenSkies (Figure 9).

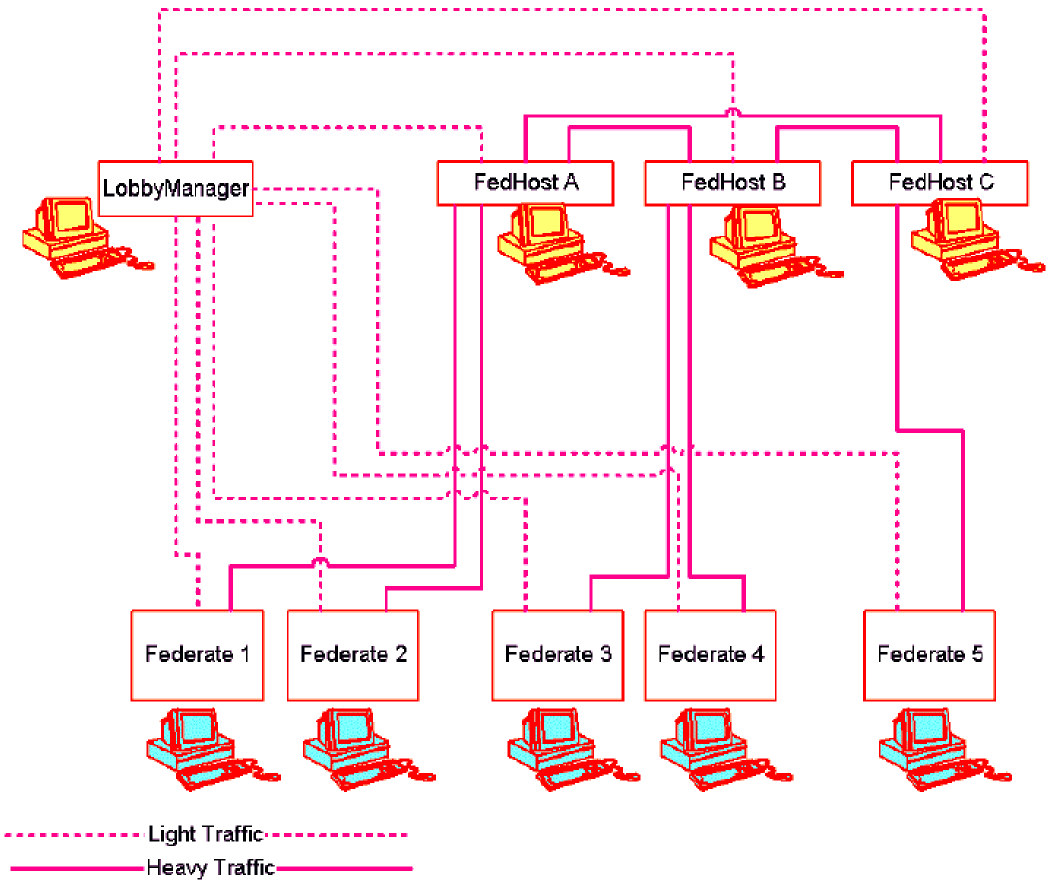


Figure 9. OpenSkies HLA-based Architecture for MMOGs (From: OpenSkies, 2009).

OpenSkies is designed to convert the peer-to-peer communications approach of typical HLA implementations to an approach similar to the client-server approach adopted by commercial games servers (Cohen et al., 2004). The development of OpenSkies is partially predicated upon the belief that the pure peer-to-peer approach of military distributed simulation will be discarded as joint military training simulations grow. Commercial game-developer acceptance of OpenSkies is not evident but the concept of alternative architectures is relevant to this thesis.

F. SH-60B MISSION REHEARSAL TACTICAL TEAM TRAINER (MRT3) AS A GAME CLIENT

ONR funded the development of the SH-60B Mission Rehearsal Tactical Team Trainer (MRT3) by Naval Air Warfare Command Training Systems Division (NAWC-TSD) to provide a PC-based training simulation for anti-submarine helicopter crews.

It was developed specifically with the idea that PC-based simulation might better enable the creation of communities of learners. It is built primarily upon commercial-off-the-shelf (COTS) hardware and software with the addition of some application-specific user-interface hardware.

MRT3 is a collection of independent applications that reflect the specific crew positions within the SH-60 helicopter (Figure 10). In addition to stand-alone use, the MRT3 has been extended to participate in military distributed simulations using both DIS and HLA. The SH-60B MRT3 has been incorporated into several distributed environments including Fleet Synthetic Training (FST) Exercises (Gallo et al., 2006).

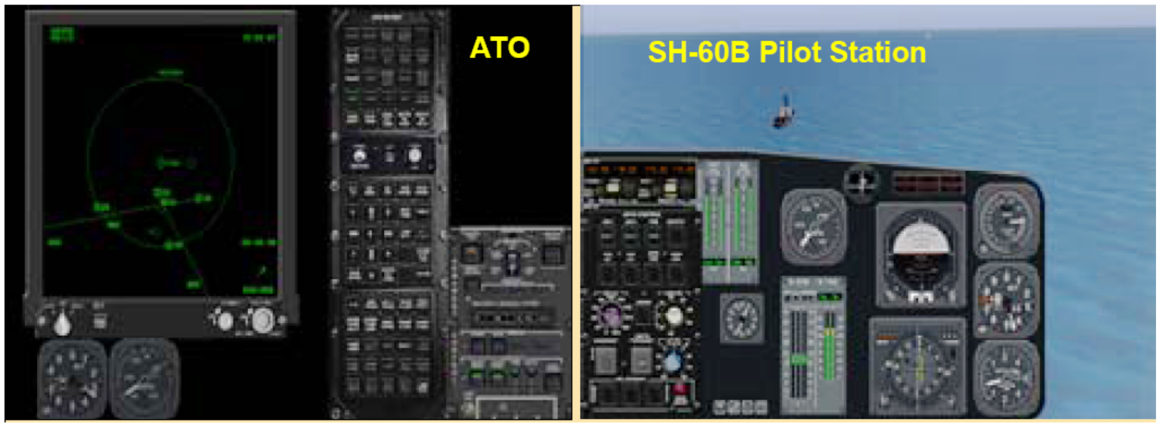


Figure 10. Screen capture from the Airborne Tactical Officer (ATO) and Pilot Station of the MRT3 (From: Gallo et al., 2006).

Since a great deal of time and effort has already been invested in developing the MRT3, and because it uses standard specifications for connecting to military DVEs, it serves as a good test case for integrating existing simulations with alternative DVE architectures.

G. X3D: OPEN SOFTWARE STANDARD FOR DEFINING AND COMMUNICATING REAL-TIME, INTERACTIVE 3D CONTENT

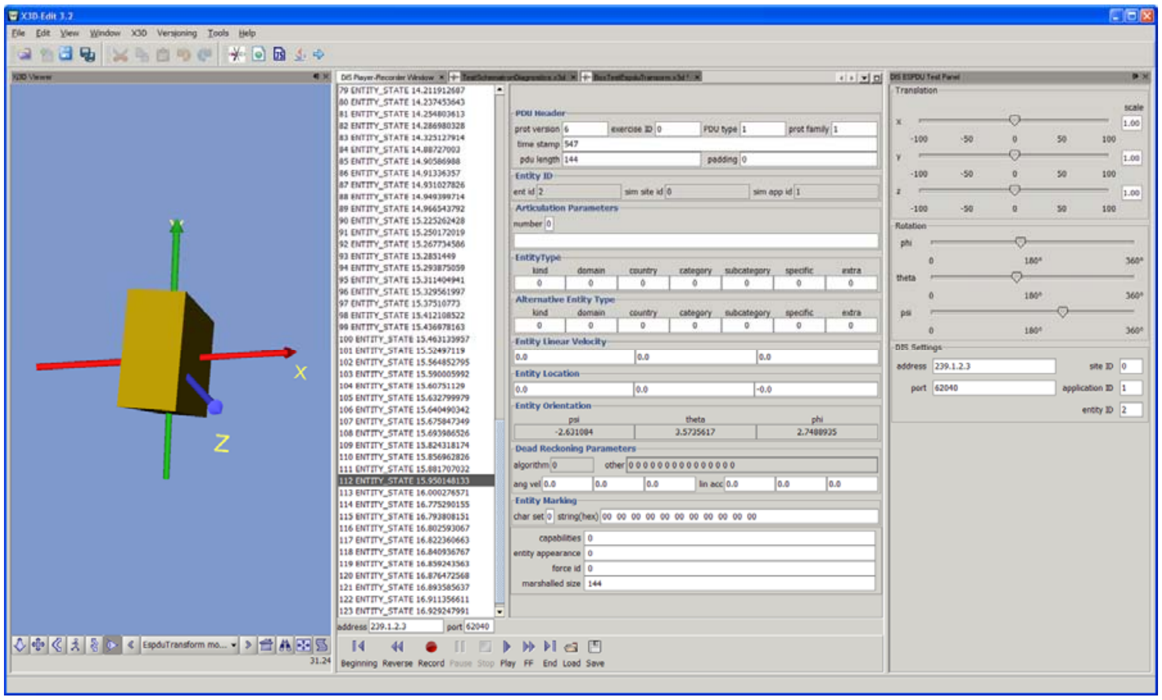
1. X3D Conceptual Framework

Despite almost three decades of graphics research, creating compelling distributed 3D virtual spaces is usually problematic due to intense competition and little interoperability between large numbers of proprietary commercial technologies. The Web3D Consortium in a nonprofit organization dedicated to the creation of open standards, specifications and best practices in order to promote the evolution of technologies that can help bring Web3D to the mainstream.

Extensible 3D (X3D) Graphics is the current Web3D standard for web-capable 3D content. It is an XML-based standard that is derived from Virtual Reality Modeling Language (VRML) that was designed to overcome several of the limitations of VRML. Since XML is a standard for describing the scene graphs of 3D content, browser and viewer developers can implement the actual rendering of these scenes in any way. Visual simulation is one of the target applications for VRML and X3D (Blais et al., 2001; Brutzman & Daly, 2007).

2. Authoring Tools

Since X3D is XML and text-based, authoring can be done in any environment. Productivity and quality depends upon the availability of user-friendly authoring tools of which there are several. A detailed survey is beyond the scope of this thesis, though numerous tools and applications are listed on the X3D Resources page (Web3D Consortium, 2009). X3D-Edit is presented as an exemplar because of its support for DIS ESPDU functionality. X3D-Edit is the authoring tool developed and maintained at the Naval Postgraduate School's Modeling Virtual Environments and Simulation (MOVES) Institute (Brutzman, 2003). One of its key tools for DVE development is the DIS Player-Recorder (Figure 11), which enables inspection and analysis of DIS PDUs as they arrive over the network.


X3D-Edit 3.2
File Edit View Window X3D Versioning Tools Help
X3D Viewer
DIS Player-Recorder Window
BoxTestEspduTransform.x3d
X
Z
EspduTransform mo...
31.24
Beginning Reverse Record Pause Stop Play FF End Load Save
DIS ESPDU Test Panel

```
79 ENTITY_STATE 14.211912687
80 ENTITY_STATE 14.237453643
81 ENTITY_STATE 14.254803613
82 ENTITY_STATE 14.286980328
83 ENTITY_STATE 14.325127914
84 ENTITY_STATE 14.88727002
85 ENTITY_STATE 14.90586988
86 ENTITY_STATE 14.91336357
87 ENTITY_STATE 14.931027826
88 ENTITY_STATE 14.949399714
89 ENTITY_STATE 14.966543792
90 ENTITY_STATE 15.225262428
91 ENTITY_STATE 15.250172019
92 ENTITY_STATE 15.267734586
93 ENTITY_STATE 15.2851449
94 ENTITY_STATE 15.293875059
95 ENTITY_STATE 15.311404941
96 ENTITY_STATE 15.329561997
97 ENTITY_STATE 15.37510773
98 ENTITY_STATE 15.412108522
99 ENTITY_STATE 15.436978163
100 ENTITY_STATE 15.463133957
101 ENTITY_STATE 15.52497119
102 ENTITY_STATE 15.564852795
103 ENTITY_STATE 15.590005992
104 ENTITY_STATE 15.60751129
105 ENTITY_STATE 15.632799979
106 ENTITY_STATE 15.640490342
107 ENTITY_STATE 15.675847349
108 ENTITY_STATE 15.693986526
109 ENTITY_STATE 15.824318174
110 ENTITY_STATE 15.856962826
111 ENTITY_STATE 15.881707032
112 ENTITY_STATE 15.950148133
113 ENTITY_STATE 16.000276571
114 ENTITY_STATE 16.775290155
115 ENTITY_STATE 16.793808151
116 ENTITY_STATE 16.802593067
117 ENTITY_STATE 16.822360663
118 ENTITY_STATE 16.840936767
119 ENTITY_STATE 16.859243563
120 ENTITY_STATE 16.876472568
121 ENTITY_STATE 16.893585637
122 ENTITY_STATE 16.911356611
123 ENTITY_STATE 16.929247991
```

address 239.1.2.3 port 62040

PDU Header
prot version 6 exercise ID 0 PDU type 1 prot family 1
time stamp 547
pdu length 144 padding 0

Entity ID
ent id 2 sim site id 0 sim app id 1

Articulation Parameters
number 0

EntityType

kind	domain	country	category	subcategory	specific	extra
0	0	0	0	0	0	0

Alternative Entity Type

kind	domain	country	category	subcategory	specific	extra
0	0	0	0	0	0	0

Entity Linear Velocity
0.0 0.0 0.0

Entity Location
0.0 0.0 -0.0

Entity Orientation

psi	theta	phi
-2.631084	3.5735617	2.7488935

Dead Reckoning Parameters
algorithm 0 other 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
ang vel 0.0 0.0 0.0 lin acc 0.0 0.0 0.0

Entity Marking
char set 0 string(hex) 00 00 00 00 00 00 00 00 00 00 00

capabilities 0
entity appearance 0
force id 0
marshalled size 144

Translation
scale
x 1.00
-100 -50 0 50 100
y 1.00
-100 -50 0 50 100
z 1.00
-100 -50 0 50 100

Rotation
phi
0 180° 360°
theta
0 180° 360°
psi
0 180° 360°

DIS Settings
address 239.1.2.3 site ID 0
port 62040 application ID 1
entity ID 2

Figure 11. DIS Player-Recorder Panel from X3D Edit 3.2

3. X3D and Distributed Virtual Environments

3D virtual scenes without interactivity do not meet the requirements of a DVE. X3D content can be remotely manipulated across the network in realtime through the use of Script nodes (Brutzman & Daly, 2007). More interestingly, the X3D specification includes the EspduTransform node (McGregor & Brutzman, 2008). This node allows distributed control over geometry nodes via DIS entity state PDUs. Visualization of the state of the entities in a DVE becomes relatively trivial if the remote applications, which control them, produce ESPDUs, which reflect their state.

4. Scenario Authoring and Visualization for Advanced Graphical Environments (Savage) X3D Model Archive

The Scenario Authoring and Visualization for Advanced Graphical Environments (Savage) archive is a collection of composable X3D models designed to enable straightforward development of web-based virtual environments. The Savage model collection is developed primarily by active-duty military students at NPS (Figure 12).

The Savage collection includes Savage Modeling and Analysis Language (SMAL) tactical metadata, which supports the 3D authoring process by embedding information about a model inside the model itself. The X3D-Edit authoring tool provides a growing number of support capabilities for building and testing composable-networked X3D models (Blais et al., 2001). Additional models can be added and used from different sources such as the Army Model Exchange (U.S. Army Virtual Targets Center, 2009).

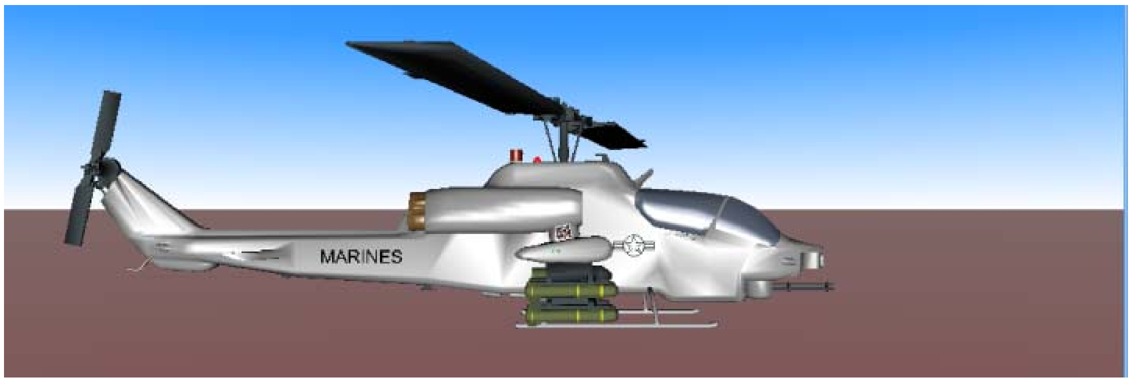


Figure 12. Model of an AH-1W Cobra developed by Maj C. B. Lakey, USMC (https://savage.nps.edu/Savage)

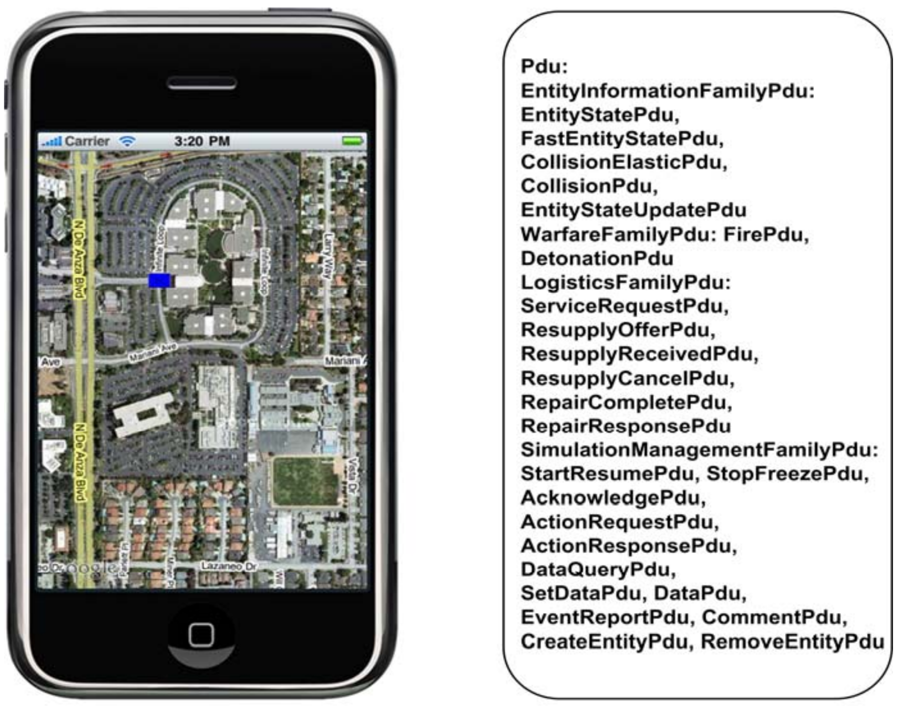
H. OPEN-DIS: OPEN-SOURCE IMPLEMENTATION OF DIS

Even more directly relevant to this work is the Open-DIS library. It is a free, open source implementation of the DIS standard implemented in C++, C# and Java. Open-DIS and SAVAGE models, combined with an application that provides game services such as Project Darkstar together provide a relatively straightforward means to implement a distributed virtual environment that may integrate DIS datastreams from multiple sources.

Open-DIS is a collection of applications and libraries to support use of DIS developed at the MOVES Institute at NPS. Its ultimate goal is to make available a full implementation of the DIS protocol in multiple programming languages for multiple platforms. It carries a Berkeley Software Distribution (BSD) license, which means that applications developed with it (or modifications to it) are not required to be released themselves as open-source (McGregor & Brutzman, 2008).

In both C++ and Java, the protocol data unit (PDU) is modeled as a class. Complete program representation of all DIS PDUs is a cumbersome task requiring over 20,000 lines of code (McGregor & Brutzman, 2008). An extensible markup language (XML) like dialect is used to structure the development of PDU classes. The Open-DIS PDU class objects are able to marshal themselves to XML or serialized Java objects. This is in addition to network reading and writing of arrays of bytes. Due to this source-code auto-generation capability the Open-DIS implementation is thus relatively lightweight. Tests have shown that up to 1,000 PDUs per second can be received and processed on commodity hardware (McGregor & Brutzman, 2008). Open-DIS is also capable of running on mobile computing devices as well (Figure 13).

Elements of the Open-DIS library form an integral part of several of the components developed in support of this thesis work. The primary advantage derived from using Open-DIS is that it was not necessary to develop an application-specific protocol. This allows the thesis work to be extended to other DIS-capable simulations relatively easily.

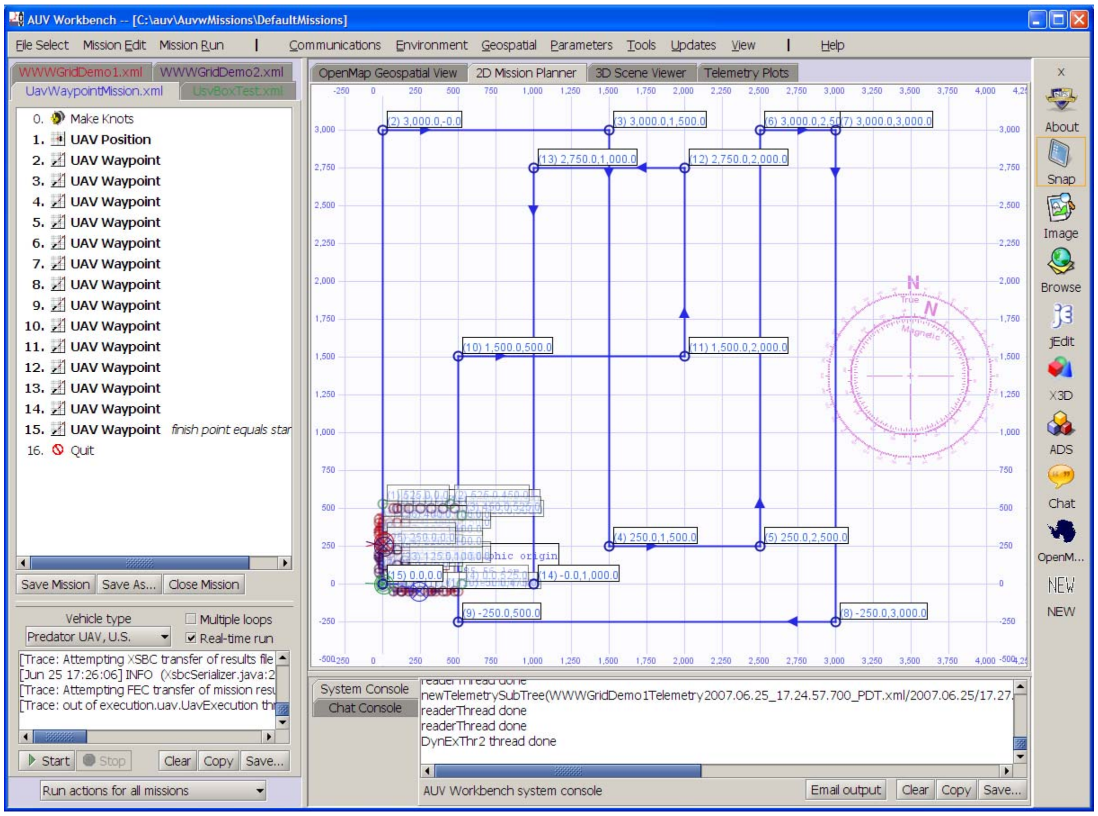


Pdu:
EntityInformationFamilyPdu:
EntityStatePdu,
FastEntityStatePdu,
CollisionElasticPdu,
CollisionPdu,
EntityStateUpdatePdu
WarfareFamilyPdu: FirePdu,
DetonationPdu
LogisticsFamilyPdu:
ServiceRequestPdu,
ResupplyOfferPdu,
ResupplyReceivedPdu,
ResupplyCancelPdu,
RepairCompletePdu,
RepairResponsePdu
SimulationManagementFamilyPdu:
StartResumePdu, StopFreezePdu,
AcknowledgePdu,
ActionRequestPdu,
ActionResponsePdu,
DataQueryPdu,
SetDataPdu, DataPdu,
EventReportPdu, CommentPdu,
CreateEntityPdu, RemoveEntityPdu

Figure 13. (l) Open-DIS running on an iPhone® controlling icon position. (r) A partial sample of the DIS PDUs implemented to date in Open-DIS.

I. AUTONOMOUS UNMANNED VEHICLE (AUV) WORKBENCH

The importance of structured data interchange for DVE interoperability is essential. Currently individual Autonomous Unmanned Vehicle (AUV) systems typically operate on their own with a proprietary data format for collected data and no command task language for mission planning that is sharable among robots. The problem of data interchange between AUVs is directly analogous to the problem of DVEs, which use heterogeneous clients (Weekley et al., 2004). The AUV Workbench (https://savage.nps.edu/AuvWorkbench) was developed at NPS to provide a tool for mission planning, mission rehearsal and mission playback (Figure 14).



0. Make Knots
1. UAV Position
2. UAV Waypoint
3. UAV Waypoint
4. UAV Waypoint
5. UAV Waypoint
6. UAV Waypoint
7. UAV Waypoint
8. UAV Waypoint
9. UAV Waypoint
10. UAV Waypoint
11. UAV Waypoint
12. UAV Waypoint
13. UAV Waypoint
14. UAV Waypoint
15. UAV Waypoint *finish point equals star*
16. Quit

Vehicle type: Predator UAV, U.S.
Multiple loops: [ ]
Real-time run: [✓]

```
[Trace: Attempting XSBC transfer of results file
[Jun 25 17:26:06] INFO  (XsbcSerializer.java:2
[Trace: Attempting FEC transfer of mission resu
[Trace: out of execution.uav.UavExecution thr
```

Run actions for all missions

```
readerThread done
newTelemetrySubTree(WWWGridDemo1Telemetry2007.06.25_17.24.57.700_PDT.xml/2007.06.25/17.27.
readerThread done
readerThread done
DynExThr2 thread done
```

AUV Workbench system console

Figure 14. Screen capture from AUV Workbench 2D view showing UAV mission waypoints.

During the execution of robot mission rehearsal or robot mission playback, AUV Workbench is able to generate ESPDUs that reflect the state of the entities whose movement behavior it is representing. It supports underwater, maritime surface, airborne, and ground vehicles. Individual vehicle state updates can be reported via classical binary DIS PDUs or as DIS-XML fragments embedded in Extensible Messaging and Presence Protocol (XMPP) chat messages. Visualizations are produced using networked X3D models from the Savage archive (Weekley et al., 2004). The AUV Workbench is therefore usable as another source of entities, which can populate a persistent DVE (Brutzman, 1994).

J. SUMMARY

Examples of the usage of large scale persistent DVEs for knowledge solicitation and distributed training have been provided. Alternatives to pure client-server architectures are being examined for MMOGs in order to reduce the tendency of the server to become a bottleneck. Since an efficient mechanism for state distribution is an essential core element for both MMOGs and large militarily useful DVEs, there are ongoing efforts to develop middleware that allows the developer to concentrate on the application instead of this core. Existing simulators and models can be integrated into a DVE that makes use of emerging technology. The Savage X3D model archive and related applications such as X3D-Edit and AUV Workbench provide a ready means to develop a DVE application using a DIS-based middleware core.

IV. METHODOLOGY

A. INTRODUCTION

Project Darkstar can be thought of as an application server for massive multiplayer online games. It is not a game implementation but a toolkit to handle the many low-level tasks that are common to all multiplayer online games. The development of a DVE application that uses the Project Darkstar toolkit has been produced in order to examine some of the questions raised in this thesis. A client simulator enables many simultaneous remote connections to be made to the server for testing purposes. Gateways are necessary to integrate simulators that do not use common standards into a common environment. Open standards are used to exchange information and visualize the state of the DVE. Performance and behavior measurement methodology is a fundamental component of a systematic development process.

B. IMPLEMENTATION OF A MMOG SERVER APPLICATION

1. Project Darkstar

a. Overview

Project Darkstar (PDS) is a free and open-source MMOG toolkit from Sun Microsystems (Project Darkstar, 2009). The server code is distributed under the GNU Public License (GPL), while the client code is distributed under the Berkeley Software Distribution (BSD) license. While both licenses are open source, this licensing choice makes modifications to the server side code “viral,” and required to be themselves open-source, while client side code can remain unconstrained. Darkstar is still officially in a pre-release form and lacks some critical release features, such as multi-server scalability. Darkstar is not yet a complete server-side implementation of game technology, but is nevertheless a highly functional toolkit that allows the server-side component of games to be constructed easily. It can be thought of as an application server for games (Figures 15 and 16).

Project Darkstar provides a framework to handle common tasks such as low latency communications, thread management, handling contention between clients for shared state, persistence, and (in future releases) scalability. Darkstar's basic insight is to treat objects on the server side as persistent, and to make state changes to them transactional and parallelizable. Darkstar provides three main services to the implementer as subsystems: a data manager, a task manager, and a channel manager (Burns, 2007; Project Darkstar, 2009).

b. Darkstar Data Manager

The data manager coordinates access to the persistent, server-side objects. The objects are stored in a database and brought into memory by the data manager; as of this writing the Berkeley database is used to store the data (Olson et al., 1999). Clients may acquire read or write access to an object, and if modified the object can be written to the database by the data manager in a transaction.

c. Darkstar Task Manager

The task manager is responsible for scheduling and running tasks, which are pieces of the server-side program that can modify object state. Tasks are assumed to be relatively short in duration; if they run for more than about 100 ms, the task manager will cancel the task, roll back any changes the task made to persistent objects, and reschedule it to run later. The assumption at that point is that the task has been delayed or blocked by some (hopefully temporarily) unavailable resource. This design choice is a product of experience in game implementations and the nature of the data manager. Project Darkstar's objective is to run as many tasks in parallel as possible. To do this, Darkstar must distribute the tasks across multiple cores, multiple CPUs, and eventually multiple hosts; if this can be done efficiently (particularly the latter) games can be called to near infinite size. For this to be practical, the tasks must be completely independent of each other, with no dependencies or communications between the tasks (Project Darkstar, 2009).

The Darkstar design therefore makes some assumptions and imposes some requirements about what a “typical” task on a server looks like. Tasks are assumed to be of short duration, either prompted by a client request or by a server-side task. The typical use case is a client either requesting a change to the state of an object or else simply requesting the state of an object, both of which can be handled in code that does not take long to run. Programmers must design and organize their tasks to conform to these assumptions in order to optimize overall performance (McGregor et al., 2009).

```
Siggi: any other way to scroll with map ecept roling buttom on mose?
Siggi: beutifull snow...
Laurentius: arrow keys...?
Laurentius: ah you mean the zoom?
Siggi: distance i mean
Siggi: yep
Laurentius: not that i know
Siggi:  is it posblie to asign a kee to thathmm
Siggi: plz :)
Laurentius: I'll add it to the list :)
Siggi: and....
Siggi: when u move mose to edg... map mowes :)
Laurentius: yes that would be nice too
Siggi: i miss that one
```

Figure 15. Call of the Kings (Gamalocus Studios) is an in-development MMORPG that uses Project Darkstar as MMOG middleware (From: Gamalocus, 2009).

When a Darkstar server fails, even ungracefully or unexpectedly, upon server reboot the task manager retrieves any incomplete or outstanding tasks from the data store for rescheduling. Persistent objects are likewise safe in the database. Thus, the power plug on a server running Darkstar can be pulled, and once rebooted the server resumes and continues as it was at the time of failure, with all object state preserved (Burns, 2007; McGregor et al., 2009).

d. Darkstar Channel Manager

All communications with the Darkstar server must be done via channels, which abstract away the housekeeping requirements of network sockets. As with HLA, Darkstar channels can define their delivery mechanism as reliable or unreliable, ordered or unordered. Generally, the Transmission Control Protocol (TCP) is used unless the channel is set to unordered and unreliable in which case User Datagram Protocol (UDP) is used. Darkstar does not define or require any particular protocol between the clients and the server. Darkstar simply passes data to the programmer-implemented protocol as a byte buffer wrapped in a Darkstar channel. On the client side, a relatively small amount of code is needed to implement the communications framework, and this can be done in any of several programming languages such as Java, C or C++, or Python (Project Darkstar, 2009). This approach is quite powerful because it allows the equivalent use of a variety of client-server data protocols if messaging semantics are consistent throughout.

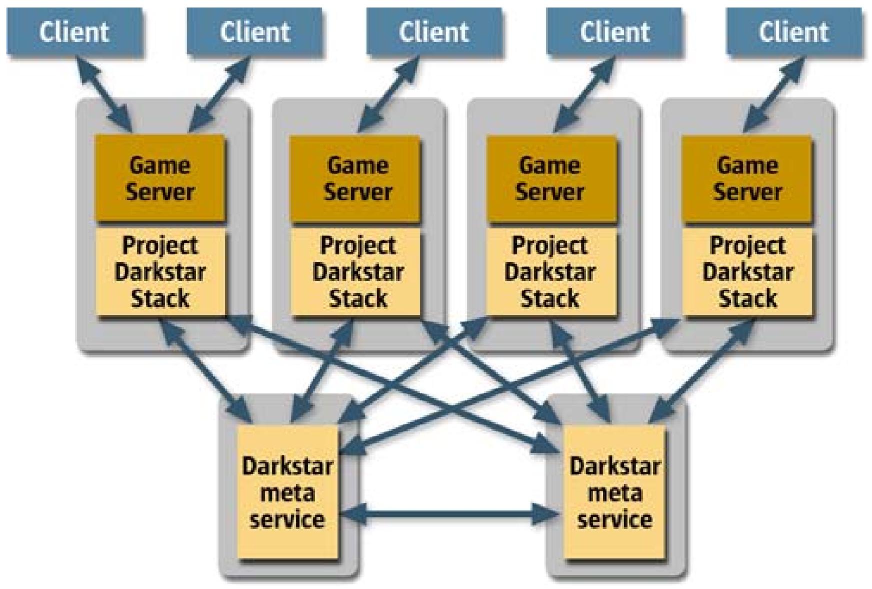


Figure 16. Project Darkstar is a Sun R&D project for the development of an application server for MMOGs.

e. Technology Roadmap

Future planned features for Darkstar include the ability to have multiple hosts that are sharing the same data managed together in an efficient way. This approach is expected to allow multiple hosts to handle client connections and, eventually, load balance Darkstar tasks across cooperating servers.

2. The MOVES Darkstar Server Architecture

As candidate middleware for MMOGs, Project Darkstar (PDS) is designed to support client-server architectures. This is the base case for the MOVES test implementation. A simple server is constructed, which allows a client to establish a connection and send messages to the server. The messages are byte buffers and DIS ESPDUs are used in this implementation. The server sends a copy of the message to every interested client. In the base case, there is no area of interest management (AOIM) and so all connected clients receive all messages (Figure 17). The improved functionality that this server adds to a typical multicast DIS environment is that much of the network programming is greatly simplified and abstracted away. DIS simulation nodes connect to the server via a gateway that listens for incoming PDUs on the network. Native PDS clients are able to connect directly to the server.

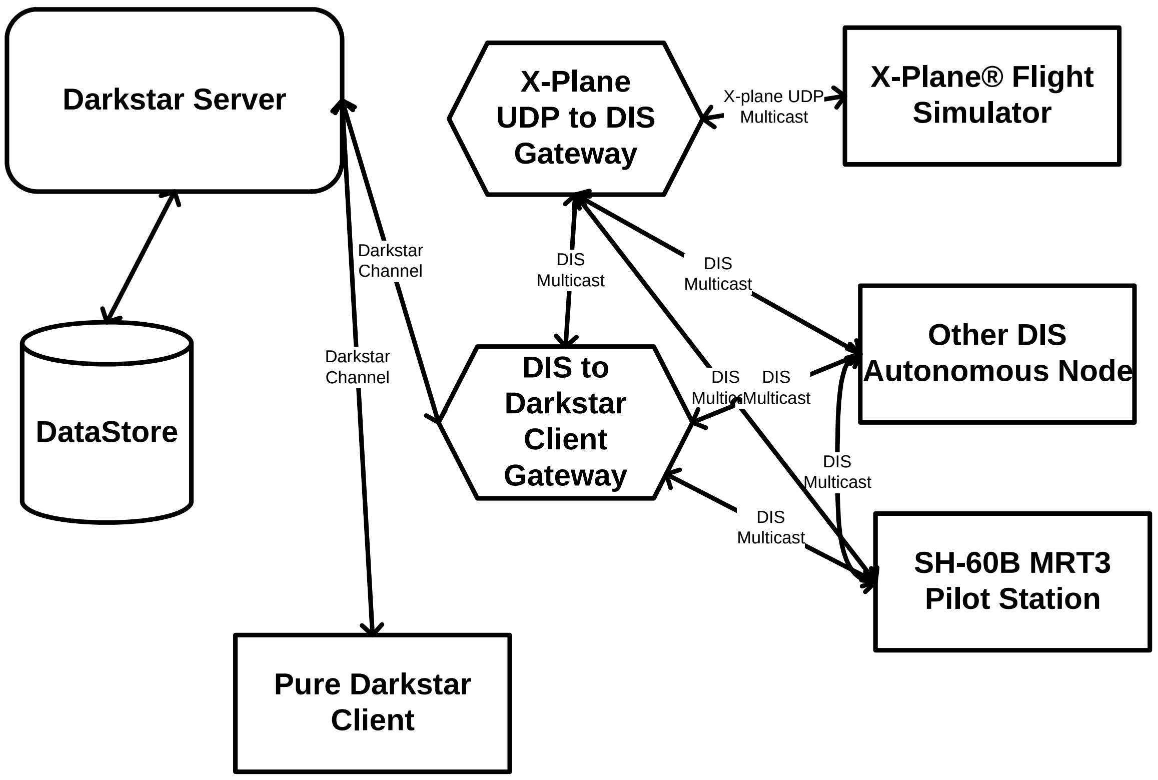


Figure 17. Overview of the architecture of the prototype DVE that includes autonomous DIS nodes and Project Darkstar clients. Gateways are used as middleware to address heterogeneity of clients.

Given this base case, an additional protocol is added next. The X-Plane simulation node uses a gateway that translates its proprietary data format into DIS ESPDUs. This approach lets all DIS-enable clients on the LAN listen to X-Plane DIS PDUs. Future work will construct a direct gateway between the X-Plane UDP and the Darkstar server.

3. Adding Server-side Game Logic

Server-side entities increase the richness of the environment and suggest the notion of an always-on and populated DVE (Figure 18). This simple representation uses the Open-DIS library and represents the server-side entity state as an ESPDU. Dead reckoning or path prediction is a DIS algorithm that is also commonly implemented in online games. Server-side dead-reckoning of both server-side entities and connected client entities has been successfully implemented in this work (Figure 19). This opens up the possibility of traffic management schemes at the server, and reconstruction of the state of other relevant clients at each connected client, using information provided by the server in the event of some fault at the client (Yasui et al., 2005).

PDS is an inherently event-based system. Careful thought must be given to which processes are to be executed as PDS tasks and which are suitably implemented as function calls responding to client requests. Yet, other tasks can sometimes be delegated to the clients themselves. Of the desirable characteristics of a DVE system, scalability and low-latency state update capabilities are particularly useful as has been discussed (Project Darkstar, 2009). Proper division and deployment of labor is essential to effective DVE design.

For this work, the base implementation of server-side logic revolves around the scheduling of so-called TickEntityTasks for every entity at some specific update rate. This task is not native to PDS but is part of the server implementation developed in support of this thesis. Darkstar tasks are either periodic or scheduled. Scheduled tasks can either be set to execute immediately or after some delay. The baseline implementation in this body of work uses a periodic task to emulate the behavior of a DIS-based entity on the server. The periodicity or tick interval is analogous to DIS heartbeat update interval. The effect of various tick intervals on the server are measured and examined in Chapter V. As stated, the middleware abstracts multi-threading away from the developer. This thesis is specifically concerned with appropriate choices of update handling when dealing with large numbers of dynamic entities.

```
Initialize (server.properties)
    Declare DataManager, ChannelManager, TaskManager

    for zero to (number of starting server entities)
        Declare an Entity

        Use the DataManager to get a reference for the Entity

        Add the Entity reference to the ScalableHashMap

        Using the TaskManager to declare a task handler
        Associate a HeartbeatTask with the reference
        Associate a TickEntityTask with the reference

    Start the ClientListener
```

Figure 18. Pseudocode of the server initialization logic. The periodicity of state updates on the server side is set at this point.

It is important to emphasize that running many parallel independent simulated entities is not the goal and is not the difficulty. Many independent threads can be executed when there is no contention between objects. The scalability challenge derives from the need to have a consistent shared state between all of the entities that make up of the environment state. The entities must be able to share knowledge if necessary. They are not independent. This is one of the fundamental distinguishing requirements for a distributed virtual environment (DVE).

The base implementation for task scheduling includes a Heartbeat Task and a TickEntityTask. The heartbeat task has a 5000 ms (5 seconds) maximum periodicity in order to satisfy the DIS standard. Any entities for which updates have not been received after five heartbeats are subsequently removed from the environment state by a Reaper Task. Future implementations may seek to eliminate the Heartbeat Task for clients that connect directly to the server since open socket connections provide an alternate mechanism for confirming that an entity is live.

The TickEntityTask is the mechanism by which dead reckoning occurs for all entities that are registered with the server. The time interval between ticks is a test point but is defaulted to 250 ms in this initial test. Dead reckoning occurs as a simple function call to a member method of the Entity object. Second-order acceleration-based dead reckoning is implemented though the default server-side entities all have acceleration equal to zero.

$$\mathbf{P} = \mathbf{P}(0) + \mathbf{U} \bullet t + \frac{1}{2} \bullet \mathbf{A} \bullet (t)^2$$

*New Entity Location = Entity Location + Entity Linear Velocity * time Step + ½ * Entity Linear Acceleration * (time Step)^2*

Figure 19. Vector and pseudocode representation of second-order motion computations used for dead reckoning. Full vector state includes both position and orientation.

Task design and implementation is important because of the default task timeout of 100 ms. Tasks that take more than this duration due to processing time or contention with shared objects are marked as failed by the middleware and rescheduled. Such failures are costly and hopefully avoided through proper overall design. Jobs that take longer periods of time to accomplish (perhaps due to computational complexity or network delays) may be better accomplished by a series of tasks that cooperatively start, monitor, and complete a long-duration job.

Simultaneous task generation creates a likelihood of object contention and the degree to which this might become a problem is examined experimentally. This implementation uses a periodic task handler to schedule a Tick Entity Task for every entity in the global state with an interval set upon server initialization. It follows that the number of tasks scheduled per second equals *Number of Entities * (1000 ms/tick interval)*. For example, 500 entities with an interval of 250 ms will theoretically result in the scheduling of 2000 Tick Entity Tasks every second. This thumb-rule does not consider other less frequent tasks that are used for server management. For instance, simple dead reckoning used here is completed in less than 0.1 ms on a 2.6 GHz x86 processor (M = 0.048 ms, SD = 0.014 ms) and so in itself does not produce task failures due to timeouts.

Task failure does occur when a task is unable to obtain a lock on the object it is associated with before the timeout deadline is reached. Such failures are easy to detect and reschedule. However, it is difficult to analytically predict when such faults occur. Study of the behavior of the system under different configurations provides some insight into such problems and also can suggest approaches for server-side state-update logic improvements. For example, allowing tasks to fail only a certain number of times or avoiding the scheduling of dependent tasks.

4. Mixed-Authority DIS Entity State Server

A mixed-authority server is simply a system in which the server maintains authority over the entities that it simulates internally while it further accepts as authoritative, state updates received from all remote clients. This is exactly in accordance with the requirements of the DIS standard (IEEE 1278). As an alternative example, server-side enforcement of consistency might be implemented by rejecting inputs from remote clients that do not correspond to the server's dead reckoned record of that entity's state. This approach, however, is a violation of the DIS assumptions. Implementing deadreckoning on the server enables an architecture by which a client-server relationship can be established between DIS nodes and the Darkstar server while maintaining the autonomy of the DIS nodes. All connections are via the server. The server creates an entity whenever a new client connects and begins to dead reckon the movement of the entity upon receipt of the first ESPDU. The client behaves as a normal DIS node except that it communicates solely with the server (Figure 20). DIS logic can be satisfactorily supported as long as server capacity and performance are sufficient.

The base implementation state distribution logic forwards every received state update to all clients. If there are *N* clients connected, theoretically this will result in *N* updates per client at the mean update rate. Although this is a naïve estimate of the traffic load at the server node, it demonstrates that such a fully interconnected state-distribution method theoretically suffers from poor scalability. For purposes of comparison, message size is considered to be equivalent to a DIS ESPDU sent as a User Datagram Protocol (UDP) packet of 172 bytes, (144 bytes payload + 28 bytes header). If there are 500 remote clients, each of which transmits one 172-byte update per second, which is forwarded by the server to every remote client, the average traffic load becomes a daunting 43 Mbytes per second.

The specific communication protocol is abstracted away from the implementation developer by the Project Darkstar middleware. Inspection of common examples shows that typically the reliable-delivery TCP is used for network traffic. When the server receives a message from the client, it compares the update to its internal model of the entity. It only forwards the message if its model does not match the update within some tolerance. This eliminates the need for transmitting expected incremental changes. When new participants log in, the server initially provides each new participant with the state of all the connected entities. Thus, all participants in the DVE share a consistent world state, even for latecomers whose presence was not known beforehand.

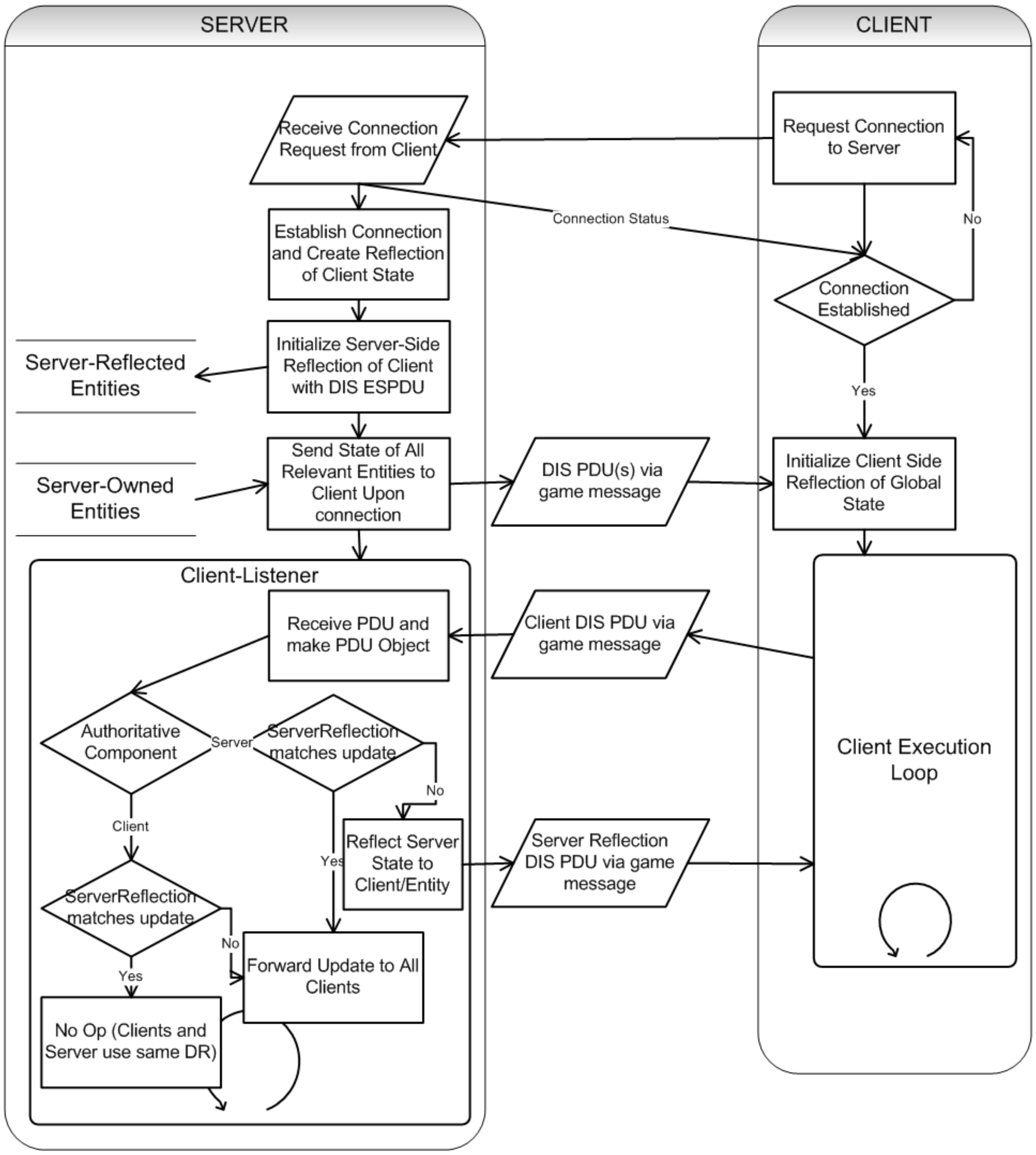

SERVER
CLIENT
Receive Connection Request from Client
Request Connection to Server
Connection Status
No
Establish Connection and Create Reflection of Client State
Connection Established
Initialize Server-Side Reflection of Client with DIS ESPDU
Server-Reflected Entities
Yes
Send State of All Relevant Entities to Client Upon connection
Server-Owned Entities
DIS PDU(s) via game message
Initialize Client Side Reflection of Global State
Client-Listener
Receive PDU and make PDU Object
Client DIS PDU via game message
Authoritative Component
Server
ServerReflection matches update
No
Client Execution Loop
Client
Yes
Reflect Server State to Client/Entity
Server Reflection DIS PDU via game message
ServerReflection matches update
No
Forward Update to All Clients
Yes
No Op (Clients and Server use same DR)

Figure 20. Flowchart showing the state exchange between the client and server for the mixed-authority DIS server. Consistent state is maintained by all existing and arriving participants.

5. Darkstar Client Simulator

A DVE system is best tested with actual distributed clients and human users. For the purpose of gaining insight into how to implement the server logic and testing the general architecture of the system, it is sufficient to use a client simulator that generates many simultaneous connections to the server and then sends messages that are analogous to what might be seen from actual clients (Kwok, 2006).

Such a basic client test application was developed as part of this effort. Platform simulation and dead reckoning behavior are added to the package by means of classes from the X-Plane to DIS gateway, which is described below. Each simulated client runs in its own thread, which is instantiated and started by the Client Simulator (Figure 21). The simulated clients attempt to establish a connection with the remote server. Once a network connection is established each enters a loop that executes its behavior.

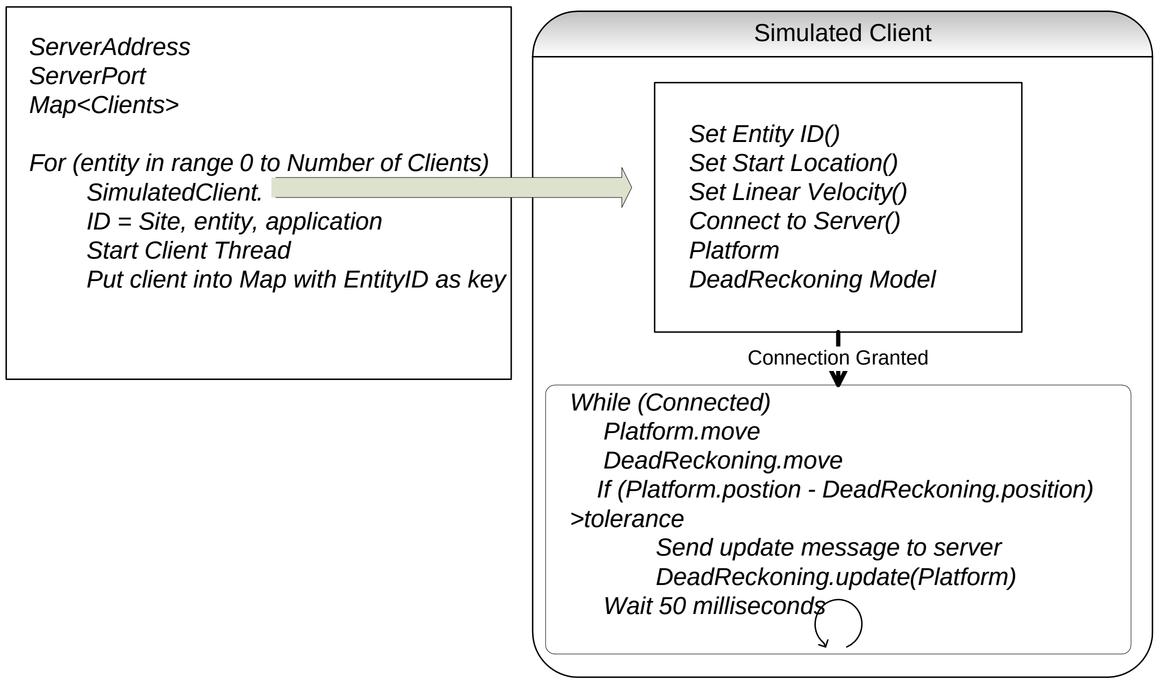


Figure 21. Diagram showing the logic of the client simulator used for testing the connection-handling behavior of the server.

The test behavior consists of three-dimensional (3D) movement at a random speed from 0 to 100 meters/second. Initial direction and location are randomly assigned. After a fixed interval, a turn is executed by setting the acceleration to a suitable value for a fixed interval. Time steps are 50 ms, which is equivalent to 20 frames (and corresponding computational updates) per second.

The simulated clients also possess a dead-reckoning model that is compared to the movement model at every time step. If the error between the two exceeds a preset tolerance distance or orientation, an update message is sent to the server via the client channel and the dead reckoning model is updated with new state values. This mimics the behavior of a remote DIS simulation node. This step is included because the only purpose of the simulator is to generate characteristic DIS traffic.

6. Project Darkstar DIS Gateway Client

Game logic in the PDS original server implementation allows the server to differentiate between entities that represent client entities and server-side entities. Game logic can then comply with DIS rules about the authority of autonomous nodes. For use in DIS multicast environments, a DIS-Gateway client is built, which listens for DIS PDUs and sends them to the PDS Server application via a connection channel. The gateway, therefore, acts as a point of aggregation, which can also segregate the DIS peer-to-peer architecture from the client-server architecture of PDS. Such a flexible scheme is quite useful for building hybrid-architecture DVEs.

C. COMPONENT GATEWAY DEVELOPMENT

To make use of existing simulations, gateways are developed as proof-of-concept of connecting heterogeneous DVEs using the middleware.

1. Laminar Research X-Plane ® to DIS Gateway

X-Plane is a popular commercial PC-based flight simulator that dominates what might be considered the hard-core PC flight simulator community. Its primary utility is the built in menus for sending messages about the state of the aircraft being represented to network-connected third-party applications. For this reason, it often used for other-than-entertainment flight simulation purposes (Laminar Research, 2009). A DIS gateway was constructed in this project in order to include X-Plane as a heterogeneous client.

There are several types of messages, which X-plane is capable of producing and these are generally described as X-Plane UDPs. The X-Plane UDPs follow a proprietary format that is described in the documentation that comes with the software (Laminar Research, 2007). The “DATA” UDP contains all of the information needed to construct a DIS ESPDU except for some details about the entity type, which can be obtained from the “SNAP” UDP.

The gateway process consists of a loop, which listens at the IP address and port to which X-plane is set to transmit. Upon receipt of an X-Plane UDP, it is interpreted and the relevant data is read into an Open-DIS ESPDU object. Since X-Plane is a real-time platform level simulation, it is capable of generating messages at very high data rates. Second-order dead reckoning with user-selectable error tolerance occurs at the gateway, and only updates which are not within this distance tolerance (as compared to the dead-reckoning model) result in the transmission of a DIS ESPDU via the user-selectable output channel. Figure 22 is a depiction of the data-handling logic of this process. Second-order dead reckoning is usually sufficient to reduce output packet rate to the heartbeat interval. Part of the launch panel for this gateway application is shown in Figure 23.

Input rates as high as 40 X-Plane UDPs per second are handled with little apparent difficulty. With dead reckoning implemented and the aircraft in X-Plane in straight and level flight, output rates are comparable to typical platform-level DIS entities.

Several opportunities for improvement exist. There is a free open-source X-Plane Software Development Kit (SDK), which can allow implementation of a more sophisticated X-Plane to DIS gateway (X-Plane SDK, 2009). More importantly, the need for gateway translation might be eliminated completely by implementing customized X-

Plane data channels directly on the Darkstar server. At that point, no translation is needed between X-plane and DIS clients, since the data channels map each protocol to consistent server-side semantics for position, orientation, velocity, accelerations, etc.

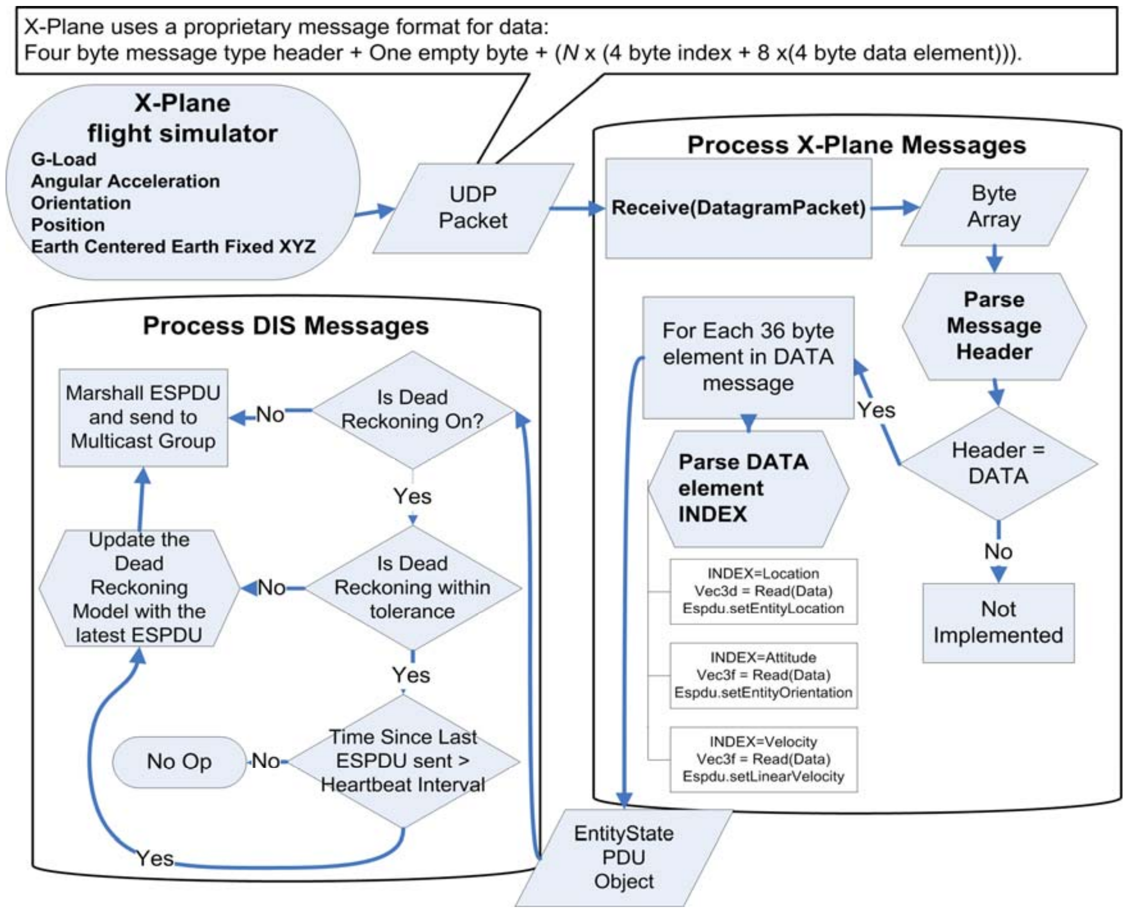
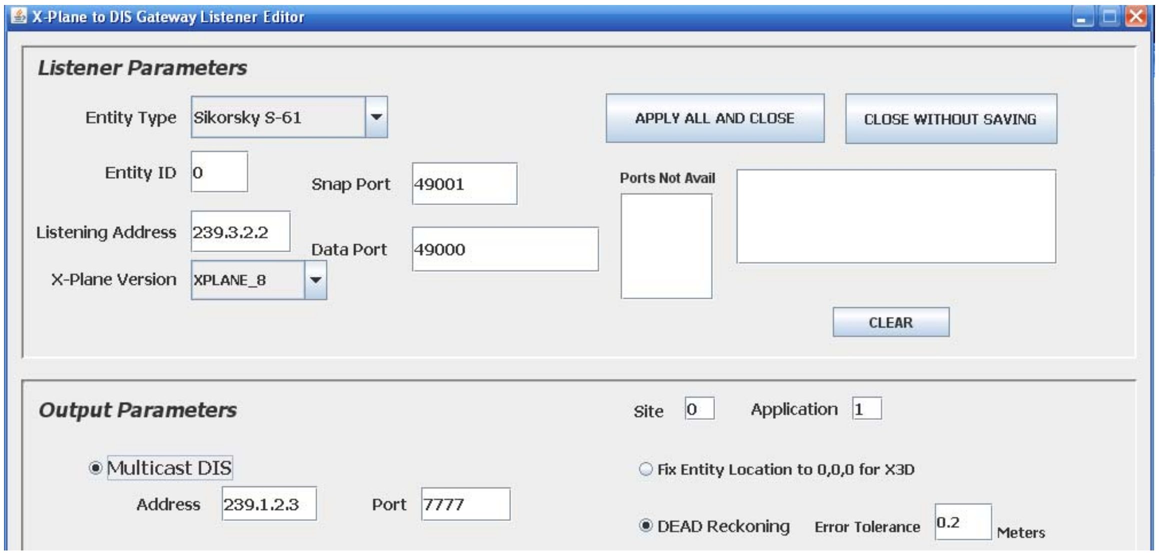


Figure 22. Diagram of the X-Plane to DIS Gateway Process Logic.


X-Plane to DIS Gateway Listener Editor

Listener Parameters

Entity Type: Sikorsky S-61
Entity ID: 0
Snap Port: 49001
Listening Address: 239.3.2.2
Data Port: 49000
X-Plane Version: XPLANE_8
Ports Not Avail

Output Parameters

Site: 0
Application: 1
(●) Multicast DIS
Address: 239.1.2.3
Port: 7777
( ) Fix Entity Location to 0,0,0 for X3D
(●) DEAD Reckoning
Error Tolerance: 0.2 Meters

Figure 23. Prototype X-Plane to DIS gateway user-interface.

2. NAVAIR SH-60B MRT3 Client

The SH-60B MRT3 already has native support for DIS built into the pilot station of the system. The HLA implementation is on the Instructor-Operator Station and it interfaces with the Common Distributed Mission Training System (CDMTS) to populate the virtual environment of the MRT3 with entities. The MRT3 is actually a small but self-contained tightly-coupled distributed virtual environment where each of the crew stations is a node that communicates with the other four nodes over a LAN.

A simple java client was developed, which listens for the ESPDUs that the pilot station multicasts, and then either forwards them to a local multicast address or else connects to the PDS DIS server and sends the message to the channel. This, unfortunately, is a one-way interaction since the MRT3 client only sends DIS packets and does not listen for DIS inputs. Full interaction with the current MRT3 system requires implementing an HLA bridge to the Darkstar server. This was not done but remains an excellent opportunity for future research. Special care must be taken, however, since independent HLA implementations are not required to support data-exchange interoperability with other HLA implementations.

D. VISUALIZATION SCENE DEVELOPMENT

1. X3D Scene Authoring

In addition to using existing simulations as clients, it is useful both for purposes of visualization and for demonstrating the potential use of X3D to provide interaction in a DVE built upon a Project Darkstar Server. One of the considerations is that DIS uses geocentric Cartesian coordinates to represent entity location. X3D uses a coordinate system oriented towards the presentation of 3D scenes on a screen. This requires a transformation at some point in the process, typically performed by the standardized X3D ESPDU transform node. In some cases, local coordinate systems are used by both X3D scenes and DIS-enabled applications.

Scene construction itself is the construction of a X3D scene-graph that contains X3D Earth terrain and entity models. The models are added as a node within an EspduTransform within the X3D scene file. The terrain and the models are used directly from the Savage model repository without modification.

2. Keyhole Markup Language (KML) Document Construction

In order to further demonstrate the utility of open standards, the Open-DIS library is used with the Document Object Model (DOM) API to generate XML documents, which represent the state of entities in the virtual environment. DOM is a supported component API of the Java API for XML Processing (JAXP). Keyhole Markup Language (KML) is an XML language focused on geographic visualization, including annotation of maps and images (OGC, 2008). Geographic visualization includes not only the presentation of graphical data on the globe, but also the control of the user's navigation in the sense of where to go and where to look.

At some user-selected periodicity, this application generates a KML document from the ESPDUs of the entities about which the client is aware (Figures 24, 25 and 26).

This allows the observation of the position of the entity in an earth-browser via the network link function. While this application is somewhat ancillary to this work, is has significant implications for the further visualization of large-scale DVEs.

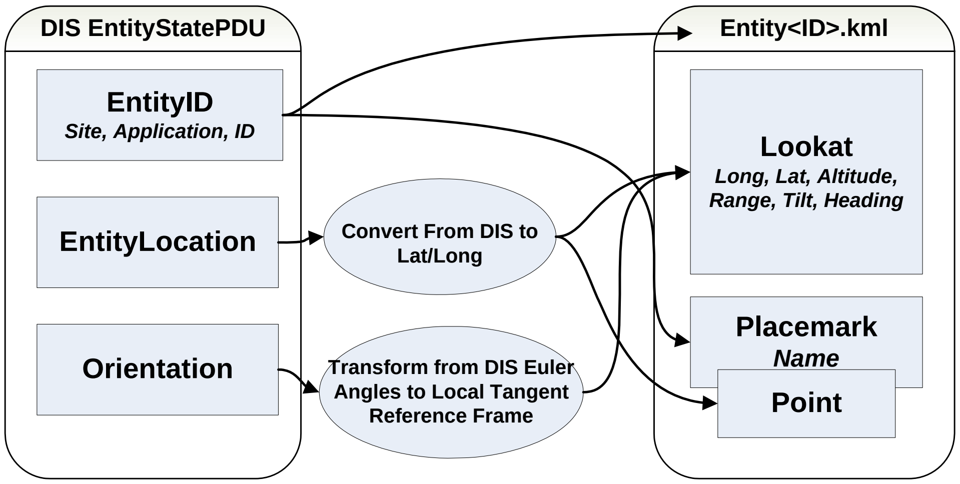
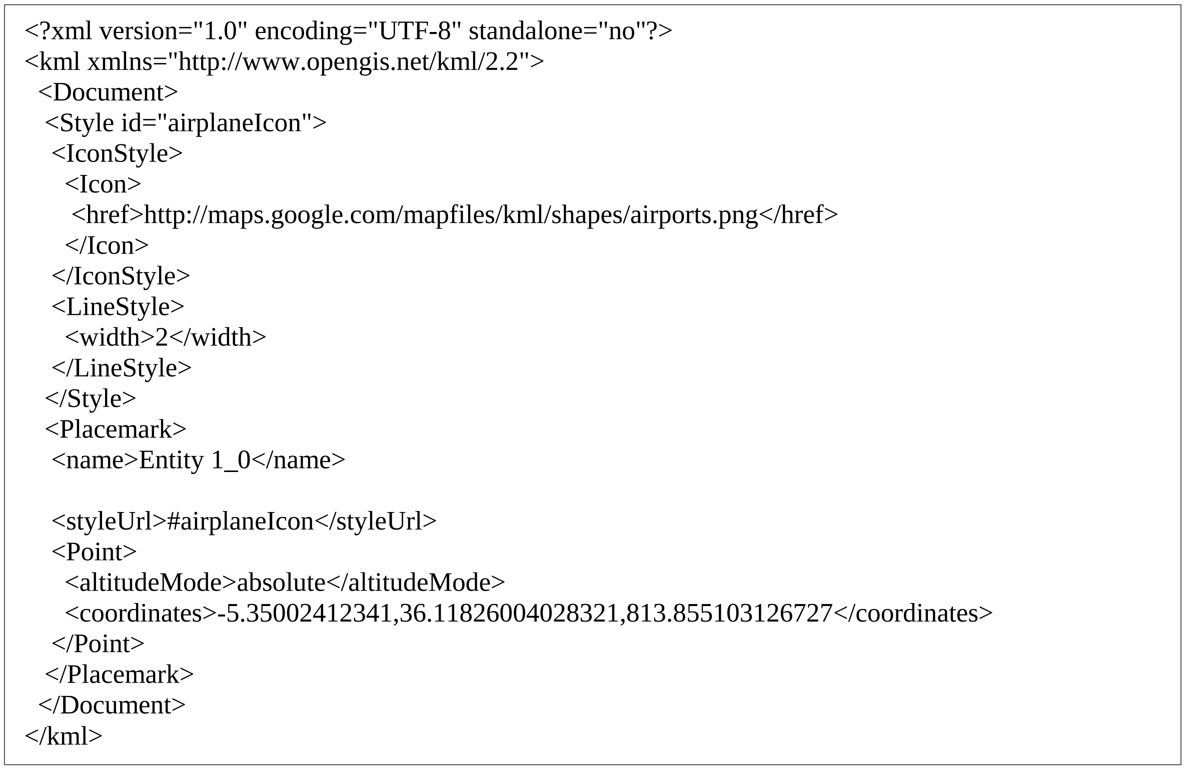

DIS EntityStatePDU
EntityID
Site, Application, ID
EntityLocation
Orientation
Convert From DIS to Lat/Long
Transform from DIS Euler Angles to Local Tangent Reference Frame
Entity<ID>.kml
Lookat
Long, Lat, Altitude, Range, Tilt, Heading
Placemark
Name
Point

Figure 24. Diagram showing data mapping from the DIS ESPDU to Open Geospatial Consortium (OGC) Keyhole Markup Language (KML) standard.



```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<kml xmlns="http://www.opengis.net/kml/2.2">
  <Document>
   <Style id="airplaneIcon">
    <IconStyle>
      <Icon>
       <href>http://maps.google.com/mapfiles/kml/shapes/airports.png</href>
      </Icon>
    </IconStyle>
    <LineStyle>
      <width>2</width>
    </LineStyle>
   </Style>
   <Placemark>
    <name>Entity 1_0</name>

    <styleUrl>#airplaneIcon</styleUrl>
    <Point>
      <altitudeMode>absolute</altitudeMode>
      <coordinates>-5.35002412341,36.11826004028321,813.855103126727</coordinates>
    </Point>
   </Placemark>
  </Document>
</kml>
```

Figure 25. Representative KML document generated by the X-Plane to DIS gateway developed in support of this thesis work.

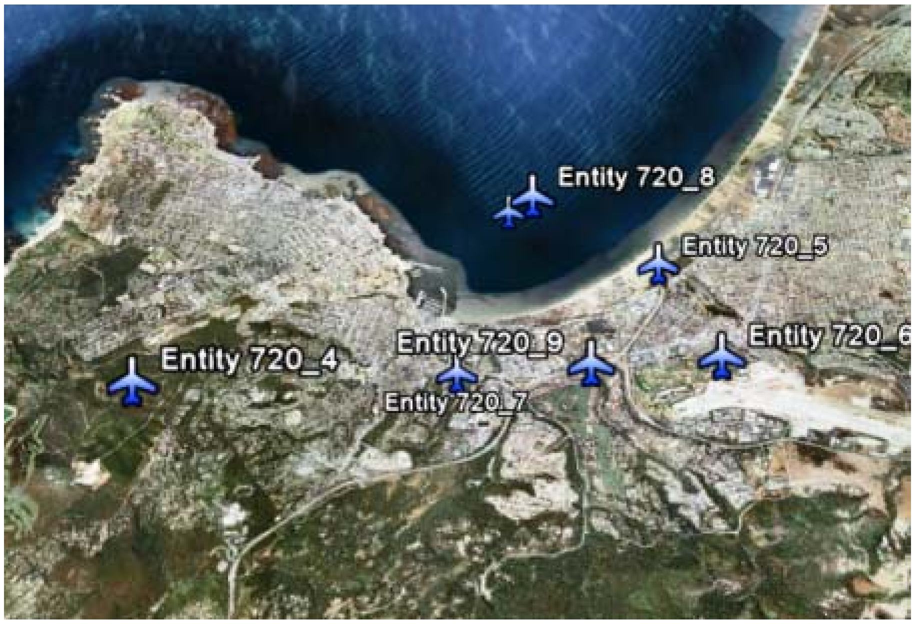


Figure 26. Screen capture showing a KML-based visualization in an Earth-browser of the entities for which the client has received state information.

E. SUMMARY

Project Darkstar is a specific implementation of middleware intended to enable MMOG/DVE deployment without the error-prone complexity of developing the communication and data-handling infrastructure. Some understanding of how the middleware operates is necessary in order to develop effective and efficient applications. The need for gateways and the importance of using open standards for interoperability are completely independent of the use of any specific MMOG/DVE middleware. Testing methodology includes monitoring the internal operation of the system and the communication among the distributed components. A successful example implementation is presented for testing purposes. Test results and analysis appear in the next chapter.

V. EXPERIMENTAL RESULTS

A. INTRODUCTION

This chapter reports details of task scheduling, metrics and measuring success/failure for various numbers of server-managed entities. This data provides interesting results and raises questions about the baseline server-side logic for updating the state of managed entities. The character of the network traffic between the server application and remote clients is also reported and explained.

B. MEASURING OVERALL SYSTEM PERFORMANCE

1. Experimental Parameters of Operation

Understanding the behavior of the server architecture and logic requires measurement under various configurations. The first study examines the effect of both tick interval and the number of server side entities on the behavior of the system. The second study examines system behavior under varying numbers of remotely connected simulated clients. The test hardware is a Dell PowerEdge 1750 rack-mounted server (Table 2).

Table 2. System Information for the Server Used in this Work

Server	Dell PowerEdge 1750 rack-mounted server
Processor(s)	Two Intel Dual-Core P4/Xeon 2.4 Ghz ea
Memory	1.0 GB
Operating System	Redhat Enterprise Linux version 4.1.2
Java Version	1.6.0_12

2. Server Profiling Test

The server profiling experimental design is (7 x 4) full factorial. Entity behavior is deterministic and no repetition is performed.

a. Independent Variables

- Simultaneous Server-Side Entities: 0, 100, 250, 500, 1000, 1500, 2000
- Tick Interval: 100, 250, 500, 750 milliseconds

b. Dependent Variables

- Failed Tick Tasks
- Successful Tick Tasks
- Entity with Tick Task Contention
- Transaction Conflict Timeouts

3. Network Traffic and Server Connection-Handling Test

Due to the need to run many simulated clients on multiple machines, all tests are conducted within a 100 Mb/s Ethernet LAN in order to avoid any routing or bridging delays. Each client generates on the order of a hundred bytes per second of traffic. However, state updates received from the server will be many times this amount, directly proportional to the number of connected clients. Since all of the simulated clients are actually connecting to the same server node this approach generates significant traffic at that node. If clients are distributed across a WAN, the requirement to operate inside of a LAN does not exist. As the tests were conducted within a LAN during periods of low traffic, the effect of network latency is not examined. The tick interval is held fixed at 250 ms and the number of server-side entities is kept at zero. The number of remote connections is varied through the set; 100, 250, 500, 750.

The client simulator application is run on two separate machines connected to the LAN by 100 Mbs Ethernet. Each is able to generate 500 connections without difficulty. The client simulator hardware platforms are as follows.

- Dual-Core x86 2.2 GHz Laptop with 2 GB RAM
- Single-Core x86 2.666 GHz Desktop 3 GB RAM

a. Independent Variables

- Remote Simulated Clients: 100, 250, 500, 750

b. Dependent Variables

- Network Traffic between Server and simulator nodes
    - Packets per second
    - Packet Size
    - Aggregate Flow
- Failed Tick Tasks
- Successful Tick Tasks
- Entity with Tick Task Contention
- Transaction Conflict Timeouts
- Successful Handle Channel Message Tasks (Part of middleware package)
- Failed Handle Channel Message Tasks

4. Measurement Methods

Data on the server's behavior at different test points is collected using a profiling plug-in to the Darkstar server (Darkstar Profiler, 2009). Coarse analysis provides insight into the suitability of this particular server implementation. A detailed examination of the tasks that are scheduled, succeed, and marked as failed points to specific implementation weaknesses. A network protocol analyzer is also used in order to characterize the communications traffic between clients and the server.

a. Profiling Data

A server profiler produced by an independent effort is used to generate a record of each test run. The record includes the following.

- Successful and Failed tasks
- Timeout exceptions
- Object contentions

The profiler package is added to the library for the Darkstar server implementation. The file darkstar-server.properties file is modified so that the profiler is started concurrently with the server (Darkstar Profiler, 2009).

The server configuration is passed is also passed by adding two properties to the darkstar-server.properties file. These are server entities and server tick interval. An experimental run consists of editing these properties using a text editor, starting the server remotely with a call to an ant target via an SSH client (PuTTY is used) (Tatham, 2009; Apache Foundation, 2009). The server is first run for one minute to allow its behavior to stabilize after initialization. After an additional two minutes, the server is stopped with a call to a different ant target. The server data store is erased so that it initializes with the next set of test parameters upon the next start. If this step is omitted, the server (which is designed for reliable persistence) will start with the configuration it had when it was stopped regardless of the darkstar-server.properties file.

The measured profiling data is then downloaded to a local machine for subsequent analysis. The profiler produces blocks of data that cover 60 seconds of run time. For tests with large numbers of entities, each recorded data block may be 15-20 MB in size.

b. Network Traffic

The Wireshark network protocol analyzer is used to capture traffic at the machines that are running the client simulators (Wireshark, 2009). The address of the server is known and so traffic from other sources is appropriately filtered from the capture trace. The aggregate of captured traffic from the two test nodes is the total communication between the server and all of the simulated clients. Since the test is done in a quiet network environment, packet loss is minimal. Background traffic at the simulator nodes is typically less than 0.1 Mb/s during the tests.

The server is started in accordance with the procedure described earlier, and then allowed to run for one minute so that it is stable and ready to accept connections. The client simulator waits one second between instantiation of simulated clients so that the server does not receive all the connection requests simultaneously, which is a reasonable precaution since its ability to handle this circumstance is not under test. After the client simulator indicates that all of its simulated clients are connected to the server, the traffic capture is initiated in the network protocol analyzer application. Five minutes of network traffic is captured to smooth out any effects of transient behavior. The capture file is saved, and the server and client simulator are each reinitialized and restarted with configuration parameters adjusted according to the test design.

C. SERVER PROFILING WITH SERVER-SIDE ENTITIES ONLY

1. Discussion

Server profile behavior with zero server-side entities reveals 189 housekeeping tasks over the 60 second period of data collection. As expected, there are no task failures or contentions at zero entities.

Curiously, the server had many more failed tasks with only 100 entities than it did at all other test levels. More usefully, at around 500 entities on the server side, the ratio of successful tick tasks to failed tick tasks reaches a maximum. Recall that the tick task represents a state update of each entity for which the server has a record. The ability of the server to maintain a consistent state is determined by the number of successful tasks. The theoretical number of generated tasks for 500 entities at an interval of 250 ms over a 60 second period is 120,000. In fact, a total of approximately 14,000 tick tasks were generated. Development of a methodology to compare the state of the server side entities after a run to an analytically predicted state is required to determine the effect of this disparity and is reserved for future work.

At 2000 server-side entities, an out-of-memory error caused the server to halt execution. No data was collected for 2000 server-side entities (Figure 27).

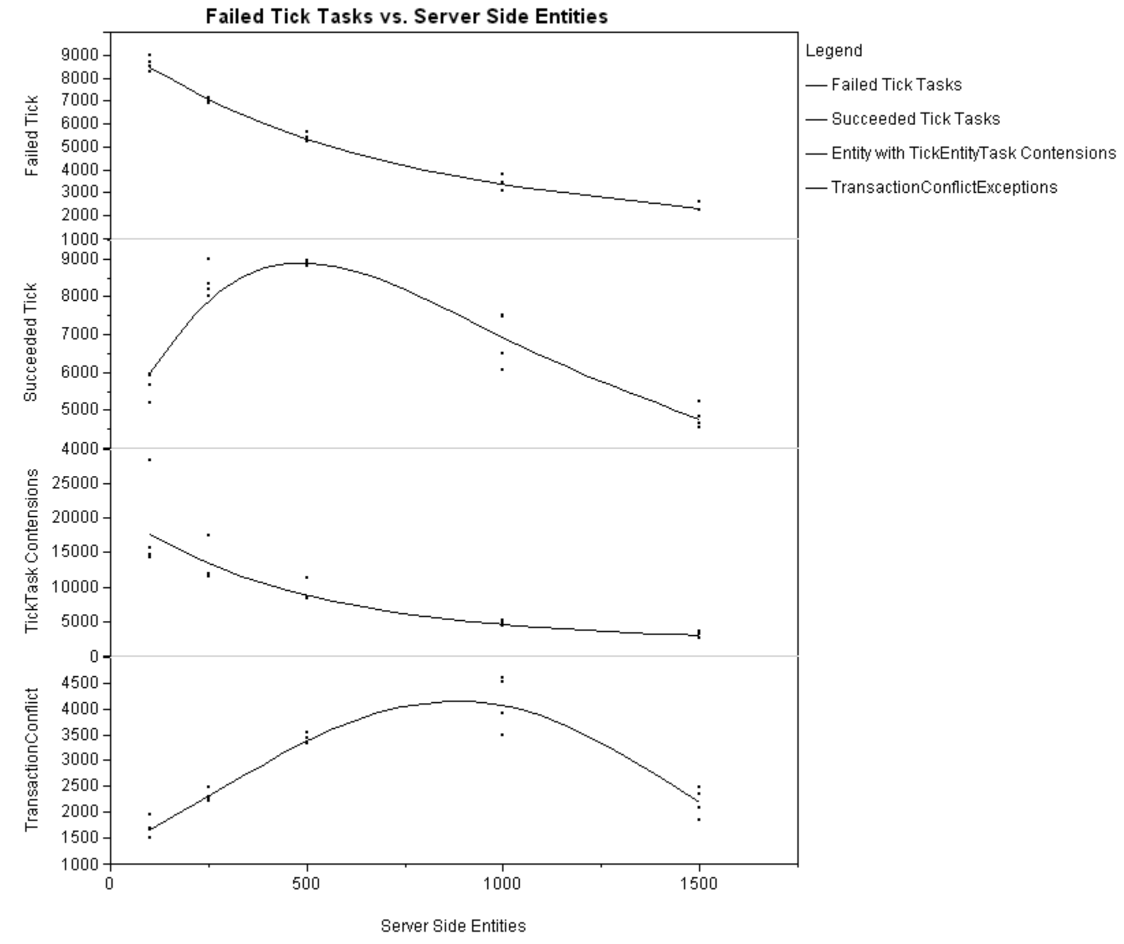


Figure 27. Effect of the number of server-side entities on task success at the server.

Tick interval had little apparent effect on the profiling data (Figure 28). Possible explanations for this measured result are that some maximum capacity is reached even at the least demanding test point or that this is a characteristic of the middleware package. Further study is required.

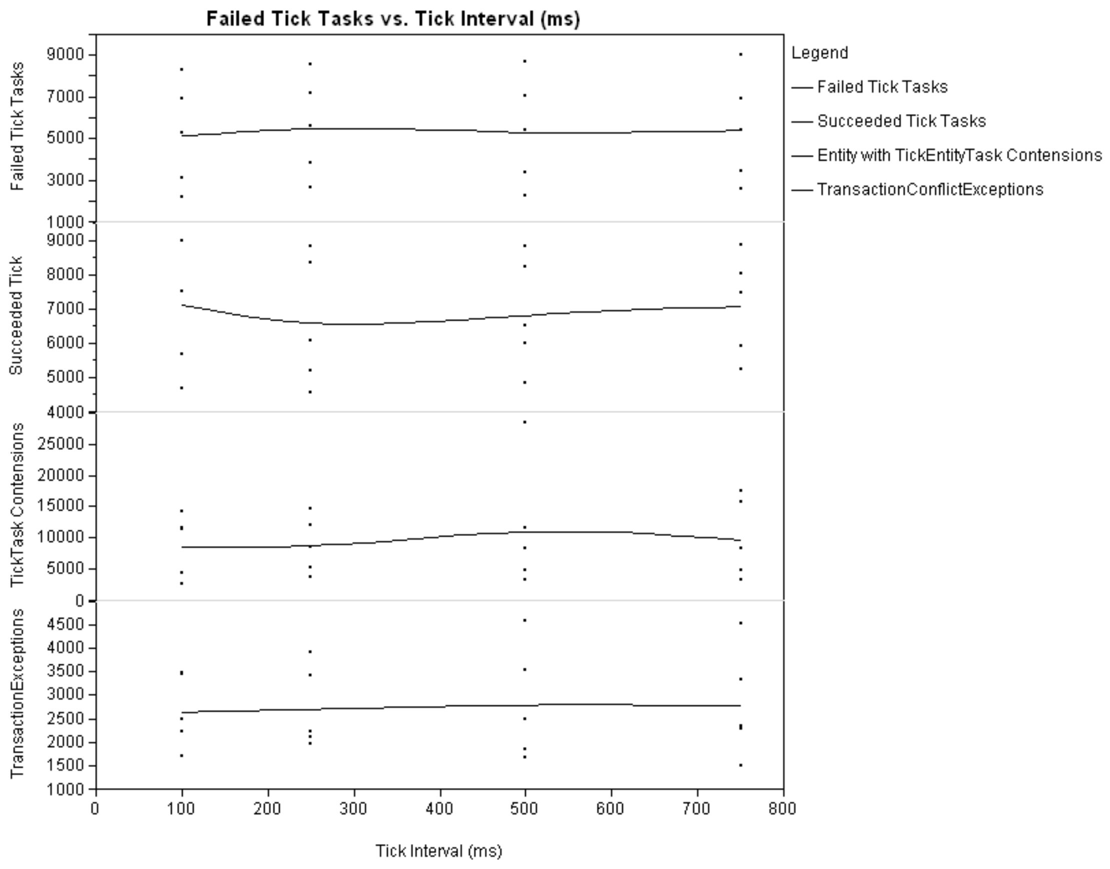


Figure 28. Effect of the tick interval on task success at the server.

D. SERVER TESTING WITH THE REMOTE CLIENT SIMULATOR

1. Discussion

Profiling of the server execution is conducted for purposes of comparison to the server-side entity only data. Since the record of each client is maintained as a server side entity, the profiling data is similar to that of the case where there are no remote clients (Figure 29).

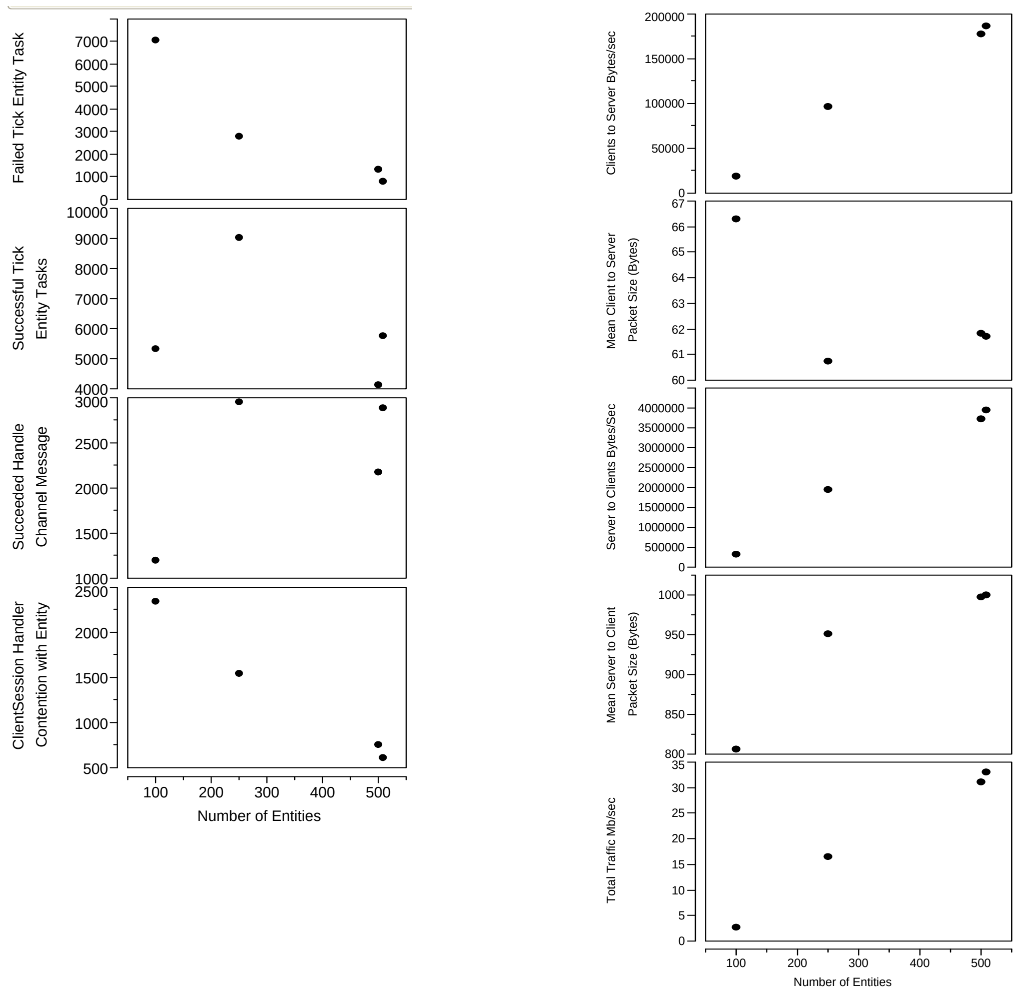


Figure 29. (l) Server profiling data showing effect of the number of connected clients. (r) Network traffic analysis showing effect of the number of connected clients on the traffic at the server.

Since all the clients are actually resident at one node, all traffic between clients and the server is between these two physical nodes. For purposes of this test, this unusual distribution of clients is not consequential. Of interest is the ability of the server implementation to handle many dynamic simultaneous connections. In addition, some idea of the network traffic generated by the implementation is obtained.

This implementation uses the Open-DIS library to marshal ESPDU objects to a byte buffer (McGregor & Brutzman, 2008). This byte buffer is the actual message sent from the server to the client using the Darkstar Channel. Although a DIS ESPDU is typically 144 bytes in length, TCP packets sent from each client to the server were exactly 54 bytes long including the protocol header of 40 bytes (Tables 3 and 4). This confirms that the message is streamed to the server rather than sent as one packet. Investigation reveals that the Project Darkstar Server uses a communication setting of TCP-no delay. This setting disables nagling, which is a TCP feature that combines several smaller packets into one larger packet (Technet, 2009).

Table 3. Summary of network traffic generated during the test.

Number of Entities	Total (Packets/Sec)	Total (Kbytes/Sec)	Clients to Server (Kbytes/sec)	Server to Clients (Kbytes/Sec)	Mean Client to Server Packet Size (Bytes)	Mean Server to Client Packet Size (Bytes)
100	681.35	338.74	18.89	319.85	66.31	806.84
250	3645.63	2057.11	96.23	1960.88	60.73	951.36
500	6612.35	3900.50	178.19	3722.30	61.84	997.67
508	6979.09	4143.78	186.39	3957.39	61.72	999.55

Table 4. Summary of per client packet rate and byte rate for server to client and client to server messages respectively.

Number of Entities	Server to Client (Packets/Second/Client)	Server to Client (Bytes/Second/Client)	Client to Server (Packets/Second/Client)	Client to Server (Bytes/Second/Client)
100	3.96	3198.55	2.84	188.93
250	8.24	7843.55	6.34	384.93
500	7.46	7444.62	5.76	356.39
508	7.79	7790.15	5.94	366.92

Despite multiple attempts including running client simulators on more than one hardware platform simultaneously, the maximum number of connected clients never exceeds 508 even though 750 clients attempted to connect. Approximately ten percent of packets sent from the server to the client simulator node are lost when 500 and 508 simulated clients are connected. No packets sent from the client simulator to the server over established channels are lost. Inspection of the profiling data does not suggest a reason for the server application’s inability to establish more than this number of connections. Inspection of plots of the traffic provides additional insight.

100 Simulated Clients:

Every simulated client executes the same behavior model. Despite efforts to stagger the initialization of the simulated clients, strong periodicity is apparent in the communication between the clients and the server. Packet rate between the clients and the server are proximate as is expected. Inspection of the throughput in Bytes/second shows that each client transmission causes a corresponding surge in traffic from the server (Figure 30). Despite a mean traffic load of 338 KB/sec, the load peaks near 1,000 KB/sec because the simulated clients appear to be synchronized. This is consistent with the server state distribution logic described earlier. The relative size of the client-to-server packet and the server-to-client packet are comparable to MMOGs discussed earlier.

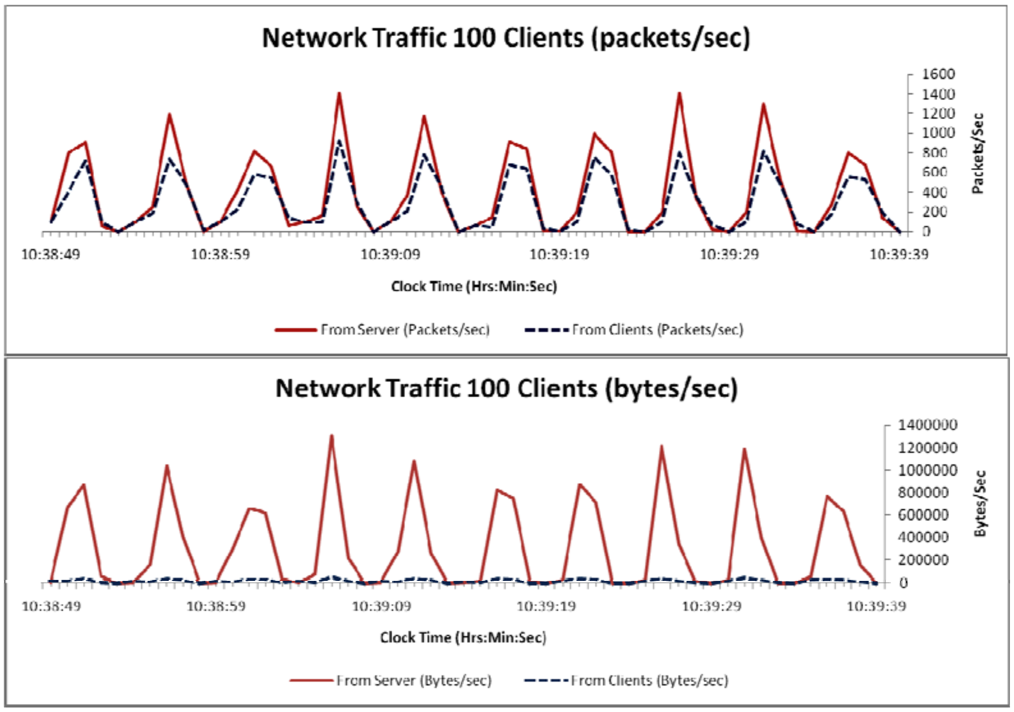


Figure 30. Plot of the network traffic between the server and 100 simulated clients. Upper plot shows packets/second whilethe lower plot shows bytes/second.

250 Simulated Clients:

While the plot of this trace tends to aggregate patterns that exist at small time resolutions, it is apparent that increasing the number of clients has the effect of smoothing out the traffic (Figure 31). This is a logical result. In this case, the peak traffic between the server and the client is only slightly above the mean of 2,057 KB/sec.

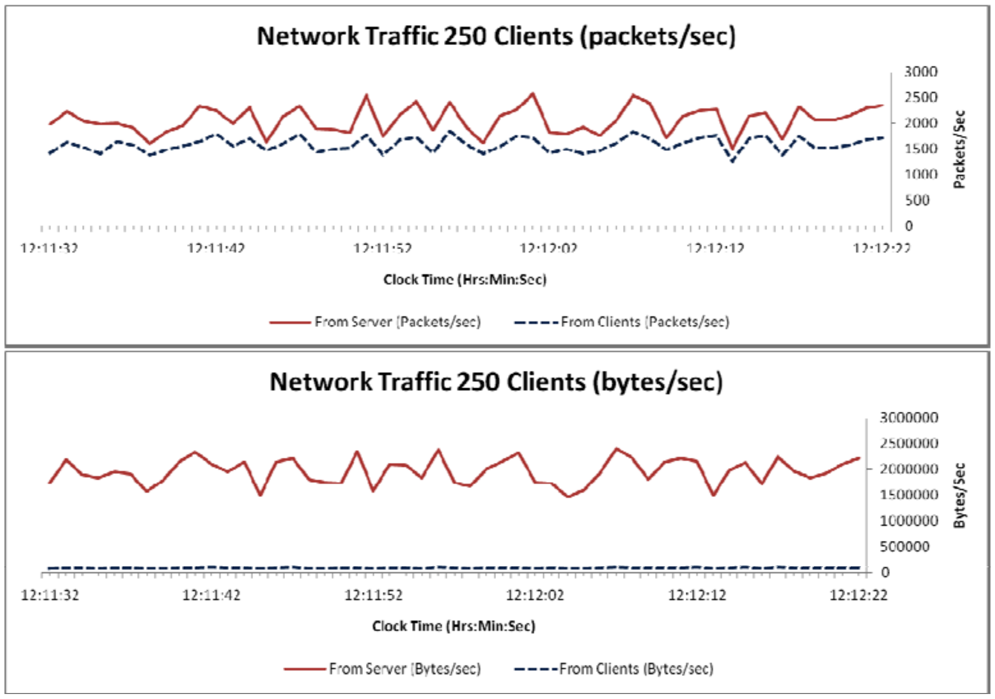


Figure 31. Plot of the network traffic between the server and 250 simulated clients. Upper plot shows packets/second whilethe lower plot shows bytes/second.

500 Simulated Clients:

The trace does not show the stable network traffic behavior that was previously observed at 250 clients in Figure 31. Ten percent of the packets sent from the server to the client simulator node are lost and the plot of the traffic resembles the familiar "saw-tooth" shape associated with rate control in TCP (Figure 32). There are no client-to-server messages lost and 90.88% of the lost packets are over 1280 bytes in length. This suggests that 500 entities is a practical upper limit to the number of simulated clients that behave similarly to connect to this particular PDS DIS server implementation.

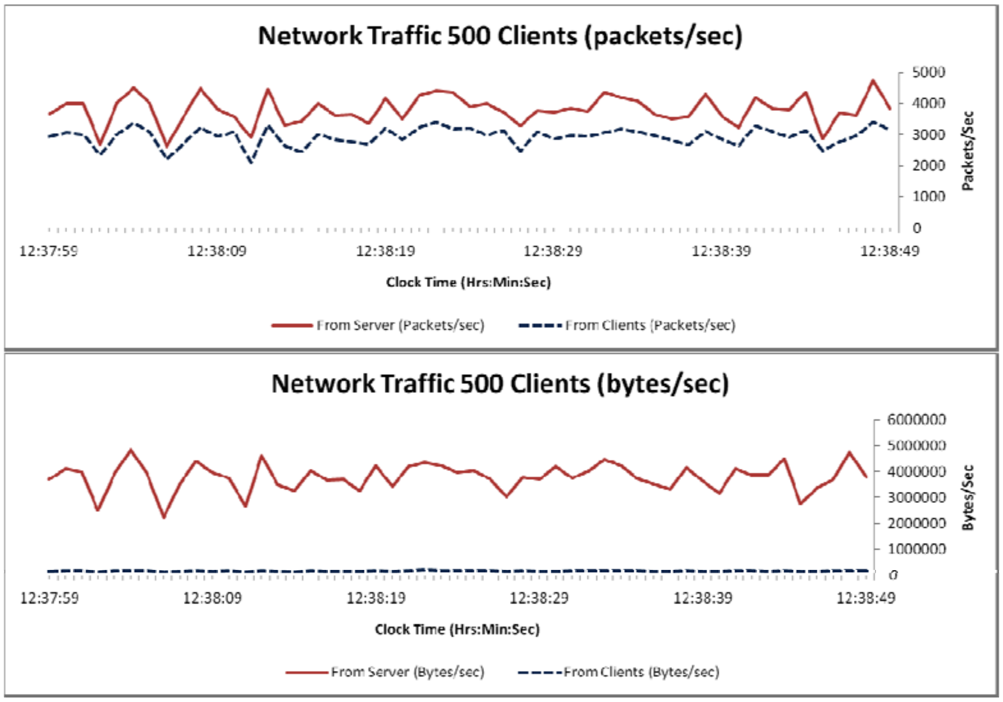


Figure 32. Plot of the network traffic between the server and 500 simulated clients. Upper plot shows packets/second whilethe lower plot shows bytes/second.

750 Simulated Clients:

The current system and test architecture are unable to exceed approximately 500 simultaneous connections. The platforms on which the client simulators are running are able to generate the required number of simulated clients however the maximum number of simultaneous connections achieved is 508. This experiment was attempted in two ways, first with 750 simulated clients on a single machine and alternatively with 375 on one machine and 375 on another. In both cases, the number of simultaneous connected clients reached exactly 508. The loss rate of packets sent from the server to the clients is 10%. No packets sent from the client simulator node to the server were lost. The plot of the traffic is qualitatively identical to the previous case of 500 clients and is not presented here.

E. SUMMARY

During the test sequence the server implementation is stable and only fails when the number of server-side entities exceeds 1,500. However, inspection of the profiling data shows that performance decreases markedly when the server must maintain state for more than 500-600 dynamic entities when object contention is possible.

The number of tick tasks generated was an order of magnitude less than expected. Not enough is understood about the middleware behavior to determine if this is acceptable. A diagnostic algorithm by which the server's model of the state might be compared against an analytically predicted state is needed to test the ability of the server implementation to maintain a consistent world state.

One particular feature of this implementation is a "Reaper Task," which removes entities from the server when no state update is received from a client for more than five "Heartbeats." During testing with either 100 or more server-side entities or 100 or more connected clients, this task failed 100% of the time. An alternative method to accomplish this behavior (or a debugged API method) is still needed.

The small size of client-to-server packets compared to the size of the DIS ESPDU that forms the body for all messages to the server introduces new questions about how the client communicates with the server (Figures 33 and 34). While one of the goals of the Project Darkstar middleware is to abstract network programming away from the developer, some knowledge about the communication behavior is required in order to use the middleware in a heterogeneous environment.

Network traffic grew geometrically with an increase in the number of connected clients from 100 to 250; however, it grew linearly when connected clients went from 250 to 500. Alternative server-side logic is likely necessary to avoid geometric growth in network traffic over certain ranges of connected clients.

The server application and test methodology described here establish a foundation for additional exploration. In addition to potential applications of this middleware, the test results themselves have generated questions that need to be answered in order to fully determine if this is a feasible approach to militarily useful distributed virtual environment state distribution and interoperability. In general, within identified limits, performance is excellent and Darkstar behaves as expected.

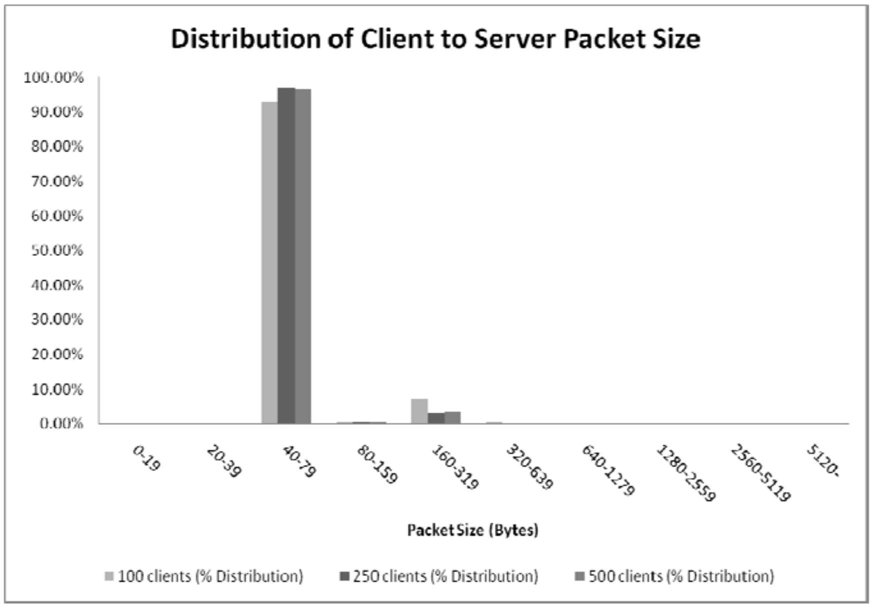


Figure 33. Packet length distribution for packets sent from the client to the server. The mean packet size was 60 bytes.

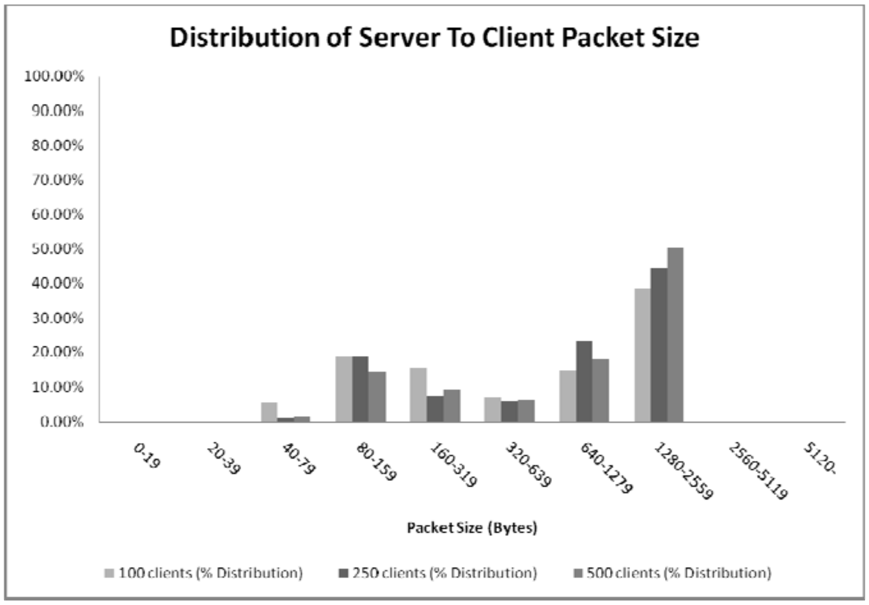


Figure 34. Packet length distribution for packets sent from the server to the client.

THIS PAGE INTENTIONALLY LEFT BLANK

VI. CONCLUSIONS AND RECOMMENDATIONS

A. TESTING CONSIDERATIONS

Since DoD distributed virtual environments must incorporate many heterogeneous simulations, and because no pure architecture can be applied with equal effectiveness across all scales, most military-useful DVEs expected to be hybrids. There is no one size fits all architecture for such diverse requirements. The growth of commercial MMOGs and the consequences for poor system performance means that significant resources are being applied toward the development of these technologies and the tools that enable them. As a result, the commercial gaming sector is likely to pass the DoD as pioneers of large-scale DVEs. One fast-progressing area where DoD developers can benefit is in leveraging the middleware that is being developed for commercial MMOG applications.

Testing reveals occasionally undesirable behavior of the baseline application used in this work and suggests alternative implementations that are expected to be more effective and more efficient. Once these core implementation issues are addressed and re-implemented satisfactorily, this work provides a ready foundation for a wide variety of important research directions. Any application requiring interactivity and shared state distribution among large numbers of remote participants is a potential avenue for applied research using these technologies.

B. CONCLUSIONS

1. Most Militarily-Useful DVEs will use Hybrid Network Architectures

The development of large-scale distributed virtual environments is a complex task. Since military operations inherently involve many heterogeneous elements, this task is made more complex for developers of militarily useful DVEs. Such difficulties are further compounded by the basic differences between simulations and C4I systems. While game developers must be concerned with heterogeneity of the hardware upon which their applications will run, military DVEs must be concerned with the syntactic, conceptual and semantic interoperability of the connected simulations. The DoD pioneered the development and use of distributed virtual environments for real-time platform-level applications with SIMNET and the Distributed Interactive Simulation protocol. Many early networked computer games borrow heavily from the conceptual architecture of DIS. As early development was for this specific application domain, it is difficult to extend to large-scale scope other than by using platform-level real-time distributed virtual environments. HLA supports composability of simulations at any level of abstraction, not just at the platform level. While the HLA standard defines the requirements for HLA compliance, the performance and interoperability of a particular system is dependent upon the commercial RTI implementation chosen. In fact, most DVEs implemented in the Defense sector are architectural hybrids as well as hybrids of HLA and DIS. This will be true for the near future.

2. MMOG Technology will Develop more Rapidly than Defense-Specific DVEs

Despite the differences between networked games and militarily useful DVEs, they share in common the primary elements of a distributed virtual environment (DVE). Large-scale militarily useful DVEs and MMOGs have analogous design requirements such as area-of-interest management and traffic management (AOIM) through position prediction. Since MMOGs are a service and not a product, they must efficiently and effectively meet these design requirements for an indefinite period. While the user-interface is a game, a MMOG system is a mechanism for distributing shared state. The significant growth of this industry creates competitive pressure to improve the technology. It is likely to develop much more rapidly than military-specific applications. The challenge is to maintain interoperability through standards while taking advantage of these rapid developments.

Persistence of a DVE is the ability to maintain the shared state for an extended period of time. A persistent environment provides an “always on” space for distributed participants to interact. It also allows the effect of participant actions to permanently affect the state of the DVE, and for the effects of participant interaction to play out over days, weeks, or even months. Unfortunately, a typical military-specific distributed virtual environment is not characterized as persistent in comparison to commercial MMOGs. A few days are the maximum reported persistence of defense-sector DVEs. Arguably, persistence is a key element in the growth of MMOGs.

3. Middleware is a Key Component of Distributed Virtual Environments

Large-scale DVE development is inherently complex. Pure client-server or peer-to-peer architectures cannot be applied uniformly across all ranges of DVE applications because networks were not designed for distributed virtual environments. Furthermore, many stand-alone systems that are usefully integrated into a DVE were not designed to operate with heterogeneous systems. Middleware is a system placed between two applications to deal with the heterogeneity of the two applications. Furthermore, middleware is placed to manage the interaction between two perhaps dissimilar applications. It is difficult to avoid the need for middleware even when strong standards are available in use. In fact, because many military DVEs are hybrids, middleware is absolutely required.

Commercial MMOGs must not only distribute state updates efficiently but they must enforce relevant consistency between clients. This must be done in a manner that is seamless to users. When the need to maintain a shared persistent environment is added, as well as requirements for authentication and failure handling, the complexity often exceeds the capabilities of most developers. Middleware that abstracts the details of this functionality from developers while providing the necessary services is a key enabler of the growth of large-scale persistent DVEs and MMOGs.

4. DVE Systems are Complex and Must be Tested to be Understood

The interaction between the system logic, the connected nodes, and the network is difficult to foresee. While there may be some insight gained from analytical models of the system, the abstractions and assumptions required to develop the model may make it difficult to extend the results to aggregate distributed behavior. The logic implemented in this case for state distribution resulted in system performance varying as the number of entities changed, but modifying the update interval has no apparent effect on the system. This result was unexpected. It is important to note that because Project Darkstar is an application middleware, this behavior may be a bug or limitation particular to this specific server implementation. Some other undiagnosed pathology might also be the cause.

System testing, which explores all of the relevant parameters, requires careful design. The experience of this work demonstrates that manual testing is not feasible for anything other than low resolution over a few design parameters. The test results do provide some insight, which may enable the development of future implementations. Furthermore, continuous persistent testing under a wide range of conditions is necessary for robust analysis of performance under expected and unexpected operational load.

C. RECOMMENDATIONS FOR FUTURE WORK

The use and value of an open-source MMOG middleware to connect heterogeneous simulators is clearly demonstrated by this work. The test results raise new questions about the performance and behavior of this particular state update and distribution architecture. The software tools developed and the lessons learned suggest several directions for further useful research.

1. Alternative Server Logic for State Distribution

The test results show that simultaneous scheduling of state update tasks for all of the entities on the server results in many failed tasks. While these initialization-failure tasks are rescheduled and ultimately completed, the recovery delays can lead to inconsistency between state updates. Additionally, there is no way to predict when a specific task will be executed. The need to distribute the processing of state updates is not peculiar to the Project Darkstar middleware. It is one of the key challenges of developing such systems. A design in which an update is scheduled for every object needs to be re-examined. Area of interest management (AOIM) and object behavior-update considerations are possible approaches to address this challenge.

The server logic has a significant effect on the network traffic produced by the system. While clients drive the behavior of the server, the response is non-linear. In fact, there is no need for the server to distribute every received state update to all clients. Prediction schemes already applied to DVEs need to be applied as consistently as possible for each type of network connection supported. The server only needs to forward updates that change the shared environment state. Alternatively, a pull versus push architecture might better be used for the server-to-client communications.

2. Consistency Comparison of System State to Distributed Client State

While inspection of the profiling data showed that failed state update tasks were rescheduled and ultimately succeeded, the number of such tasks was an order of magnitude lower than expected. This raises the question as to whether or not object state is being updated as expected. Without comparison to some expected state, this is difficult to examine.

The states of the entities on the server, which represent distributed clients need to be compared to an analytical model of the simulation environment as well as the actual clients themselves. Distributed simulation state consistency characterization is a nascent research area specifically with regard to DVEs. The ability to characterize distributed consistent state formally is an important for systematic development of useful operational systems.

3. Two-Way Interaction for Legacy Simulation Clients via an MMOG Middleware

The steps necessary to connect simulation clients to an MMOG middleware are discussed earlier. The extent of implementation in this work did not go beyond one way communication from the client to the other clients and the server, although it permits visualization and measurement. This approach is incomplete. To achieve a true DVE such communications must be two-way. In order to implement this, the client-specific communication protocols must be reconstructed from the standard DIS protocol used for communication in this system. Any necessary coordinate transformations must be included as well.

Since many legacy platform simulations are designed for updates at essentially display frame-rate, the low rate of updates from this MMOG-like system may require message spawning at some legacy clients. No assumptions about the ability of the legacy system to handle updates at a fraction of the rate it was designed for ought to be made, so interface adapters may be needed.

The legacy simulation itself must have the ability to discover objects from the messages it receives; otherwise, all such interacting objects will need to be manually inserted and a communication channel assigned to it. This may be, in fact, desirable if only certain distributed objects are of interest. Once again, the need is clear to achieve both syntactic and semantic interoperability when establishing interoperability among dissimilar systems.

4. X3D and JAVA MMOG Framework for Militarily Useful DVE.

Two of the major challenges in broadly expanding the use of DVEs for military applications are the development and distribution of client user interfaces that support network demands. Almost all DVE applications require the installation of proprietary client software for participation. Information technology management policies often make this licensing requirement untenable for government use. Furthermore, the models developed for these proprietary clients are generally not usable for other applications due to their closed nature, lack of documented testing and possible encumbrance by hidden patents.

Since X3D is the Web3D Consortium standard for web-based 3D graphics, tools and models developed for any application are readily transported to any other application. The only requirement to interact with a scene is a suitable X3D browser. Several dozen X3D browser implementations and importers are currently available including open-source versions and commercially supported software packages. Each of these approaches is compatible with X3D.

X3D animation can be controlled by embedded networked scripts and also directly by DIS entity state PDUs. While the traffic at the network server node may be high when there are many simultaneous participants, the traffic at distributed clients can be relatively low. Three Kilobits per second (Kbps) is typical for such systems and comparable to the client node traffic for MMORPGs. This enables distributed simulation across even very low-bandwidth networks provided the client nodes are widely distributed.

5. Consistency Enforcement for Distributed Simulation

Commercial distributed game service providers often use a client-server connection model in order to control the quality of the game-play experience. This is important because users will not maintain their paid subscriptions if the experience is poor. For military applications of DVEs, credible interactions are necessary for effective participant immersion. Armored vehicles that move as fast as aircraft or ignore the effects of weapons fire (for example) reduce the face validity of the system. For commercial games, various forms of cheating are one of the greatest disruptors of game-play experience. It is assumed that any client may be dishonest. The opposite assumption holds for developers of military-application DVEs. Even when all clients are honest, differences between (or weaknesses within) specific simulations may have the same effect as cheating, without the malicious intent.

Consistency enforcement is the active assurance of a shared state between the participants of a DVE. Although already difficult to implement in a game, the heterogeneous nature of military DVEs may greatly increase this difficulty because knowledge about the internal logic and external behavior of the client is not always known. Game developers control all nodes; developers of heterogeneous DVEs do not.

A basic form of consistency enforcement includes control of interactions but not the movement of the client entity. This is analogous to hybrid DVEs, which distribute position updates in a peer-to-peer fashion and interactions between entities in a client-server fashion. The idea is that the movement only changes the shared environment state, for that entity and for others, which can see that entity.

a. Army Research Laboratory (ARL) DIS Lethality Server

An example of consistency enforcement in an otherwise-autonomous node DVE is the DIS Lethality Communication Server developed by the Army Research Laboratory (ARL). In this approach, a single vulnerability/lethality server provides standard DIS damage states to entities. The system is designed to centralize the maintenance of computing damage state stables; thus, simplifying the configuration of DIS exercises. One key goal is to eliminate variability in lethality results. Including such a server as a federate in an HLA federation is a logical consequence (Sauerborn, 1999). There is no conceptual difference between that server and the system developed for this work. Both are means of distributing a shared consistent state.

b. NAVAIR UAV/UCAV Distributed Simulation Infrastructure

Naval Air Systems Command (NAVAIR) has developed an unmanned aerial vehicle (UAV)/unmanned combat aerial vehicle (UCAV) distributed simulation infrastructure. Its objective is to explore how UAVs and UCAVs can function together in the future. The element of the infrastructure that is relevant to this work is the use of a weapons-effect server in a DIS-based DVE (Twesme, 2003).

A potential application of MMOG technology for simulated UAV operations is as a persistent data-store. Real-world objects or environments of interest under observation by UAVs might be updated over days, months, and years in a persistent virtual environment.

6. Migrating MOVES MMOG Server to NPS Hamming High Performance Computing Cluster

The current implementation runs on a single server within the LAN and 500 simultaneous connections and up to 1500 dynamic server-side entities were demonstrated. Alternative state distribution logic may enable higher numbers but they will still be on the order of 1000 clients per server. The Naval Postgraduate School recently installed the *hamming* system for high-performance cluster computing. The *hamming* system is a Sun Microsystems 6048 blade system consisting of 144 blades that together contain a total of 1152 cores. The system has 112 terabytes of disk space (Naval Postgraduate School, 2009). This system provides an opportunity for the development of an NPS or DoD wide persistent virtual environment. Ten percent of this capability could theoretically support on the order of 10,000 simultaneous participants. Recall that the user interface is defined at the client level. Such participants are not necessarily individuals using a computer to control an avatar in a 3D scene but could be any distributed device or individual, which has a need to share a common state.

7. Construction of a Bridge between an HLA Federation and a MMOG Server

One of the strengths of HLA is its conceptual support for simulation composability. More importantly, HLA is the declared standard of distributed simulation implementation in the DoD. A militarily useful DVE must be capable of participating in HLA federations. The feasibility of composing a MMOG middleware application into an HLA federation is an important research direction. The OpenSkies middleware discussed earlier is an effort in this area. The open-source Portico implementation of an RTI is also worth exploring (Portico, 2009).

8. MMOG Application as a Track Data Conversion and Distribution Hub for Command and Control

An MMOG is essentially a service for distributing and maintaining shared state information of a virtual environment. This is directly analogous to command and control systems, which distribute and maintain a virtual copy of the real-world state. The consistency of that state can never be better or timelier than the quality of the real-world information, which feeds it.

A logical application of the state-distribution mechanism of a MMOG middleware is to provide a means for distributing track data for command and control. The word “game” in MMOG may suggest that this is a ridiculous notion. Recall, however, that underneath the game implementation, large modern MMOGs are among the highest-performing distributed systems with several having achieved over 50,000 distinct simultaneous participants (IGDA, 2004). While the consequences for system failure cannot compare to the reliability requirements of military systems, they are not inconsequential. In any event, the technical challenges are similar.

A basic demonstration of the utility of a track distribution hub includes the capability to convert data from one format to another. This sees the system as a form of middleware in addition to being a state distribution mechanism providing DIS mappings to and from other C4I languages is a good start.

Further work can also connect hand-held devices such as GPS-capable cell phones. As with most scalable technologies, the most important success metric is the ease of integration rather than the sophistication of the implementation.

9. Establish Persistent Darkstar MMOG Virtual Battlespace

Since commercial online games now have lifetimes measured in years, without interruption, a similar capability can be achieved for military DVEs. This effort can address the significant limitations inherent in typically transient, ephemeral military simulations. Ongoing measurement, testing, expansion and improvement will lead to further capabilities and lessons learned. Establishing public and controlled-access DVEs for simulations based on existing model archives such as Savage and SavageDefense using X3D Earth as a backdrop is a relevant goal.

10. Integrating Effective Shared Physics Governing Sensor Interactions

Entities in military DVEs not only need to have physically based motion governing their travel through virtual space, but also need high-resolution physics for sensor applications. This is essentially true for naval scenarios where the determination of sensor propagation may be computationally expensive and require durations of many seconds to complete. Additional work is needed to accomplish consistent coherent shared physics for phenomenology other than motion in DVEs.

APPENDIX. SOURCE CODE

Source code developed in support of this thesis is available at URL: http://open-dis.sourceforge.net/Open-DIS.html

Specific excerpts of source code are presented below.

```
public void initialize(Properties props)
  {
  logger.log(Level.INFO, "Started Persistent Virtual World");

  // Manages persistent objects
  DataManager dataManager = AppContext.getDataManager();

  // Manages communications channels
  ChannelManager channelManager = AppContext.getChannelManager();

  // Periodic tasks to run
  TaskManager taskManager = AppContext.getTaskManager();

  try
  {
    // Retrieve an existing, known entity from the data manager.
    //initialize() is called only the very first time
    // a datastore is created, when the datastore will always be empty.
    Entity anEntity = (Entity)dataManager.getBinding("ENTITY(0,0,1)");
    logger.log(Level.INFO, "Found existing entity object");
  }
  catch(NameNotBoundException nnbe)
  {
    // We have a brand new object store that is completely empty. Add some objects and tasks to it.
    logger.log(Level.INFO, "Creating initial server-side entities and tasks");

    // The hash map contains a key of a string in the format of
    // ENTITY(x,y,z), using the entityID of the DIS entity,
    entitiesMapRef = dataManager.createReference(new ScalableHashMap());
    ScalableHashMap entitiesMap = entitiesMapRef.get();

    // Save the hash map of entities in the system under a named string
    dataManager.setBinding("ENTITIES,” entitiesMap);

    //Get the server side entity count from the properties file
    try{
    serverSideEntityCount = Integer.parseInt(props.getProperty("server.entities"));
    }catch (NumberFormatException nfe){
     System.out.println("Unable to Parse server.entities value using default of " + serverSideEntityCount);
    }//end catch
```

Source code excerpt showing server initialization code of the Darkstar Server implementation used for this work.

```
   for(int idx = 0; idx < serverSideEntityCount; idx++)
   {
    Entity anEntity = new Entity();
    anEntity.lastPduReceived.getEntityID().setSite(0);
    anEntity.lastPduReceived.getEntityID().setApplication(0);
    anEntity.lastPduReceived.getEntityID().setEntity(idx);

    anEntity.currentState.getEntityID().setSite(0);
    anEntity.currentState.getEntityID().setApplication(0);
    anEntity.currentState.getEntityID().setEntity(idx);

    //Set a random initial velocity and position for each entity with max in any dimension
    anEntity.setEntityRandomVelocity();
    anEntity.setRandomEntityStartLocation();

    anEntity.controlLocation = Entity.ControlLocation.SERVER;
    ManagedReference<Entity> anEntityRef = dataManager.createReference(anEntity);

    // Add other information, such as entity type. We really should be reading
    // info from a config file.

    // Put the entity into a list of entities. Should this be a reference being added?

    entitiesMap.put(anEntity.toString(), anEntity);

    // Instruct the scheduler to run a heartbeat task for this entity, which
    // will send out an ESPDU at a given interval on the DIS channel.

    // We set the task to contain a handle to the task
    // itself. If the underlying entity object is deleted--perhaps because it hasn't
    // been heard from, or some other reason--the task will cancel itself when it discovers
    // the object missing in the task's run() method.

    HeartbeatTask heartbeatTask = new HeartbeatTask(anEntityRef);
    PeriodicTaskHandle heartbeatHandle = taskManager.schedulePeriodicTask(heartbeatTask, 10000,
HEARTBEAT_FREQUENCY);
    heartbeatTask.setTaskHandle(heartbeatHandle);

    // Tick frequency. Mostly a test of object contention. The tick method in an
    // entity gets called every DEFAULT_TICK_INTERVAL ms.
    // A task-per-entity is a somewhat suspect choice as you go to more and
    // more entities. It might be better to have one tick task that simply
    // cycles through the list of all entities. However, calling tick()
    // on all the entities may take a while, longer than the task scheduler
    // is willing to give. So it's a tradeoff.

    //Get the tick Interval from the properties
    try{

    tickInterval = Long.parseLong(props.getProperty("server.tickInterval"));

    }catch (NumberFormatException nfe){
    System.out.println("Unable to Parse server.tickInterval value using default of " + tickInterval);
     }//end catch

    TickEntityTask tickEntityTask = new TickEntityTask(anEntityRef);
    PeriodicTaskHandle tickTaskHandle = taskManager.schedulePeriodicTask(tickEntityTask, 10000, tickInterval);
```

Code excerpt showing task scheduling in the Darkstar server implementation.

```
switch(dataType)
{
  case FRAME_RATE:
   dataReader.skipBytes(32);
   break;

  case ANGULAR_VELOCITY:
   Vector3Float angVel = deadReckoningParameter.getEntityAngularVelocity();
   angVel.setX(dataReader.readFloat());
   angVel.setY(dataReader.readFloat());
   angVel.setZ(dataReader.readFloat());

   //Set the angular velocity dead reckoning parameter
   deadReckoningParameter.setEntityAngularVelocity(angVel);

   dataReader.skipBytes(20); // 12 bytes read, 32 bytes in group, 20 left to read
   break;

  case ATTITUDE:
   Orientation att = espdu.getEntityOrientation();

  att.setTheta((float)Math.toRadians(dataReader.readFloat()));
  att.setPhi((float)Math.toRadians(dataReader.readFloat()));
  att.setPsi((float)Math.toRadians(dataReader.readFloat()));

  dataReader.skipBytes(20); // 12 bytes read, 20 bytes left

   //Actually set the espdu to the decoded values
   espdu.setEntityOrientation(att);

   break;

  case LOCATION:
   Vector3Double loc = espdu.getEntityLocation();
   loc.setY((double) dataReader.readFloat());
   loc.setX((double) dataReader.readFloat());
   loc.setZ(FEET_TO_METERS*(double) dataReader.readFloat());

   loc.convertLatitudeLongitudeAltitudeToDis();

   if (DEBUG) System.out.println("x=" + loc.getX() + ,”" + loc.getY());
   dataReader.skipBytes(20); // 12 bytes read, 20 bytes left

   espdu.setEntityLocation(loc);
   break;

  case VELOCITY:
   Vector3Float linearVel = espdu.getEntityLinearVelocity();
   dataReader.skipBytes(12);

   linearVel.setY(dataReader.readFloat());
   linearVel.setZ(dataReader.readFloat());
   linearVel.setX(dataReader.readFloat());

   dataReader.skipBytes(8); // 12 + 12 = 24 bytes read, 8 left out of 32

   espdu.setEntityLinearVelocity(linearVel);
   break;
```

X-Plane to DIS Gateway: data packet parsing code excerpt.

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF REFERENCES

Apache Software Foundation. ANT. Retrieved April 15, 2009, from, http://ant.apache.org/

Beigbeder, T., Coughlan, R., Lusher, C., Plunkett, J., Agu, E., & Claypool, M. (2004). The effects of loss and latency on user performance in unreal tournament 2003®, *Proceedings of ACM SIGCOMM '04 Workshops, Portland, Oregon*.

Bettner, P., & Terrano, M. (2001). 1500 archers on a 28.8: Network programming in age of empires and beyond. Retrieved June 15, 2009, from http://www.gamasutra.com/

Blais, C., & Brutzman, D. (2001). Web-based 3D technology for scenario authoring and visualization. *Proceedings of I/ITSEC 2001*.

Boer, C., Bruin, A., & Verbraeck, A. (2006). Distributed simulation in industry–A survey part 1–The COTS vendors. *Proceedings of the 2006 Winter Simulation Conference*.

Bogojevic, S., & Kazemzadeh, M. (2003). The architecture of massive multiplayer online games. Master's Thesis. *Department of Computer Science, Lund Institute of Technology*.

Bonk, C. J., & Dennen, V. P. (2005). Massive multiplayer online gaming: A research framework for military training and education. Retrieved March 12, 2009, from http://oai.dtic.mil/oai/oai?verb=getRecord&metadataPrefix=html&identifier=ADA431271

Britton, C., & Bye, P. (2004). *IT architectures and middleware* (2nd ed.). Addison-Wesley, Boston.

Brutzman, D. (1994). *A virtual world for an autonomous underwater vehicle*. PhD Dissertation, Naval Postgraduate School.

Brutzman, D. (2003). X3D-edit authoring for extensible 3D (X3D) graphics. *ACM SIGGRAPH 2003 Educators Program*. San Diego, California. Course notes, examples, Retrieved September 15, 2009, from http://x3dgraphics.com

Brutzman, D., & Daly, L. (2007). *Extensible 3D graphics for web authors.* Morgan Kauffman, San Francisco.

Brutzman, D., Zyda, M., Pullen, J. M., & Morse, K. L. (2002). Extensible Modeling and Simulation Framework (XMSF): Challenges for web-based modeling and simulation. Findings and Recommendations Report of the XMSF Technical Challenges Workshop and Strategic Opportunities Symposium.

Burns, B. (2007). Darkstar: The java game server. *O'Reilly Media*. Retrieved March 22, 2009, from http://oreilly.com/catalog/9780596514846/index.html

Ceranowicz, A. Z., Torpey, M., Hellfinstine, W., Evans, J., & Hines J. (2002). Reflections on building the joint experimental federation, *Proceedings of the 2002 I/ITSEC Conference*, Orlando, Florida.

Chassot, C., Loze, A., Garcia, F., Dairaine, L., & Cardenas, L. R. (1999). Specification and realization of the QoS required by a distributed interactive simulation application in a new generation internet. *Lecture Notes in Computer Science.* Berlin: Springer, 75–91.

Chen, K., Huang, P., & Lei, C. (2006). How sensitive are online gamers to network quality? *Communications of the ACM*, *49*, 11.

Chen, K., Huang, P., Huang, C., & Lei, C. (2005). Game traffic analysis: an MMORPG perspective. *Proceedings of the international Workshop on Network and Operating Systems Support for Digital Audio and Video*. Stevenson, Washington.

Cheung, S. E., & Loper, M. L. (1994). Traffic characterization of manned-simulators and computer generated forces in DIS exercises. *Fifth Annual Conference on AI, Simulation and Planning in High Autonomy Systems*. Gainesville, Florida.

Choi, S., & Tan, H. Z. (2004). Effect of update on perceived instability of virtual haptic texture. *Proceedings of 2004 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Sendai, Japan.

Chwif, L., Barretto, M. R. P., & Paul, R. J. (2000). On simulation model complexity. *Proceedings of the 2000 Winter Simulation Conference*. Orlando, Florida.

Claypool, M. (2005). The effect of latency on user performance in Real-Time Strategy games, Computer Networks, *49*(1), *Networking Issues in Entertainment Computing*, 52-70.

Cohen, C. J., Buse, R. C., Haanpaa, D., & Jacobus, C. J. (2004). From HLA to MMOG and back again, *Simulator Interoperability Working Group 2004 Fall Workshop*.

Cybernet. (2009). Retrieved May 20, 2009, from http://www.openskies.net

Dahmann, J. S. (1999). The high level architecture and beyond: Technology challenges. *Proceedings of the Thirteenth Workshop on Parallel and Distributed Simulation*. Atlanta, Georgia.

Darkstar Profiler. (2009). Retrieved August 15, 2009, from http://code.google.com/p/darkstar-profiler/

Davis, P. K., & Anderson, R. H. (2003). *Improving the composability of DoD models and simulations*. RAND, National Defense Research Institute, Santa Monica, California.

Deering, C. (2009). AAA game titles need to cost more. Retrieved August 10, 2009, from http://www.gameguru.in/general/2009/03/aaa-game-titles-need-to-cost-more-says-chris-deering/

Diehl, S. (2001). *Distributed virtual worlds*. Berlin: Springer-Verlag.

DIS Steering Committee. (1994). *The DIS vision: A map to the future of distributed simulation, technical report. IST-ST-94-01*. Institute for Simulation and Training, University of Central Florida.

Emilsson, K., Fannar, H., Jonsson, K., Matthiasson, J., Woodward, M., & Wyld, J. (2009). Infinite space: An argument for single-sharded architecture in MMOs. *Game Developer*, *16*(6), 15–20.

Feng, W., Brandt, D., and Saha, D. (2007). A long-term study of a popular MMORPG. *Proceeding of Netgames '07*, Melbourne, Australia.

Feng, W., Chang, F., Feng, W., & Walpole, J. (2005). A traffic characterization of popular on-line games. *IEEE/ACM Transactions on Networking*, *13*(3), 488–500.

Fowler, P., & Levine, L. (1993). *A conceptual framework for software technology transition*. CMU/SEI-93-TR-31. Software Engineering Institute.

Fritsch, T., Ritter, H., & Schiller, J. (2005). The effect of latency and network limitations on MMORPGS: A field study of everquest2. *Proceedings of NetGames '05*. New York: New York.

Fujinoki, H. (2006). On the support for heterogeneity in networked virtual environment. *Proceedings of Netgames '06*. Singapore.

Gallo, A. W., Glass, J. P., Frye, C., Douglass, R., Velez, C., Buckley, J., & Cipolla, L. (2006). ASW VAST MRT3: The tip of the virtual spear. *Proceedings of the Interservice/Industry Training, Simulation and Education Conference 2006.*

Gamalocus. (2009). Call of the kings. Retrieved September 10, 2009, from http://www.gamalocus.com/

Garlan, D., Allen, R., & Ockerbloom, J. (1995). Architectural mismatch or why it's hard to build systems out of existing parts. *Proceedings of the 17th international Conference on Software Engineering.*

Glinka, F., Ploβ, A., Müller-Iden, J., & Gorlatch, S. (2007). RTF: A real-time framework for developing scalable multiplayer online games. *Proceedings of Netgames '07.*

Hall, R., & Novak, J. (2008). *Online game development*. Delmar, Clifton Park, New York.

Hsu, C-c., Ling, J., Li, Q., & Kuo, C. C. (2003). The design of multiplayer online video game systems. *Proceedings of the SPIE*, 5241, 180–191.

IEEE 1278.1A-1998. (1998). Standard for Distributed Interactive Simulation-Application protocols. Retrieved July 12, 2009, from http://standards.ieee.org/reading/ieee/std_public/new_desc/compsim/1278.1a-1998.html

IEEE 1516-2000. (2000). Standard for modeling and simulation (M&S) high level architecture (HLA)–framework and rules." *IEEE Std 1516-2000.* Retrieved June 10, 2009, from http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=893287&isnumber=19334

IETF-MMOX. Retrieved September 4, 2009, from https://www.ietf.org/mailman/listinfo/mmox

IGDA Alternate Reality Games Special Interest Group. (2006). Alternate reality games white paper. Retrieved August 20, 2009, from http://www.igda.org/arg/whitepaper.html

IGDA Online Games Special Interest Group. (2004). Persistent worlds whitepaper. Retrieved April 10, 2009, from http://www.igda.org/online/IGDA_PSW_Whitepaper_2004.pdf

Jardine, J. L. (2008). *The hybrid game architecture: Distributing bandwidth for MMOGs while maintaining central control*. Master's Thesis. Brigham Young University.

Jensen, G. (2008). Presentation: Draft Concept in support of Massive Multiplayer Online Wargame Leveraging the Internet. Office of Naval Research (ONR). Presented at Naval Postgraduate School, December 2008.

Johnson, M., Guili, R., Oberg, S., Beebe, M., Ford, R., & Shockley, J. (2004). Integration of CCTT and JCATS in an LVC exercise. *Euro Simulator Interoperability Working Group Winter Workshop, 2004*.

Keune, C. M., & Coppock, D. (1995). Synthetic Theater of War-Europe (STOW-E) Technical Analysis. NCCOSC RDT&E Division. San Diego, California.

Kim, B., Johnson, R., Youssef, R., Vallerand, A. L., Herdman, C., Gamble, M., Lavoie, R., Kurts, D., & Gladstone, K. (2005). JSMARTS Initiative: Advanced Distributed Simulation across the Government of Canada, Academia and Industry–Technical Description. Defence R&D Canada, TM 2005–101.

Knutsson, B., Lu, H., Xu, W., & Hopkins, B. (2004). Peer-to-peer support for massively multiplayer games. *Proceedings of IEEE Infocom 2004*.

Kuhl F., Weatherly R., & Dahmann, J. (1999). *Creating computer simulation systems: An introduction to the high level architecture*, Prentice Hall PTR.

Kuiper, P. J., & Lemmers, A. J. (2000). Opening up full-mission flight simulation for networked simulation exercises. *AIAA Modeling and Simulation Technologies Conference*, Denver, Colorado.

Kumar, S., Chhugani, J., Kim, C., Kim, D., Nguyen, A., Dubey, P., Bienia, C., & Kim, Y. (2008, September). Second life and the new generation of virtual worlds. *IEEE Computer Society Journal*.

Kwok, M. (2006). *Performance analysis of distributed virtual environments*, PhD Dissertation, University of Waterloo, Canada.

Laminar Research. (2007). *X-Plane 8.64 manual*. Columbia, South Carolina.

Laminar Research. (2009). *X-Plane*. Retrieved February 10, 2009, from http://www.x-plane.com/

Lecky-Thompson, G. (2009). *Fundamentals of network game development*. Course Technology, Boston.

Lenoir, T. (2003). Programming theaters of war: Gamemakers as soldiers. In Robert Latham (Ed.), *Bombs and bANDWIDTH: The emerging relationship between IT and security* (pp. 175–198). New York: New Press.

Lent, M. V. (2008). The business of fun. *IEEE Computer*, *41*(2), 101–103.

Lindley, C. A. (2003). Game taxonomies: A high level framework for game analysis and design. Retrieved August 8, 2009, from https://www.gamasutra.com/features/20031003/lindley_pfv.htm

Lu, F., Parkin, S., & Morgan, G. (2006). Load balancing for massively multiplayer online games. *Proceedings of Netgames '06*, Singapore.

Macedonia, M. R. (1995). A network software architecture for large scale virtual environments. PhD Dissertation. Naval Postgraduate School.

Macedonia, M. R., & Zyda, M. (1997). A taxonomy for networked virtual environments, *IEEE Multimedia*, *4*(1), 48–56.

McGregor, D., & Brutzman, D. (2008). Open-DIS: An open source implementation of the DIS protocol for C++ and Java. *Simulator Interoperability Working Group (SISO) Fall Workshop, 2008*.

McGregor, D., Rashid, T., & Brutzman, D. (2009). Integrating DIS with a massively multiplayer online game engine. *Simulator Interoperability Working Group (SISO) Fall Workshop, 2009.*

McVearry, K., Birman, K., Freedman, D., van Renesse, R., & Weatherspoon, H. (2008). RAPID: RAPID prototyping in Distributed Mission Operations (DMO) environments. AFRL-RI-RS-TR-2008-93. Odyssey Research Associates.

Microsoft. (2009). Pastry: A substrate for peer-to-peer applications. Retrieved September 20, 2009, from http://research.microsoft.com/en-us/um/people/antr/PASTRY

Miller, D. C., & Thorpe, J. A. (1995). SIMNET: The advent of simulator networking. *Proceedings of the IEEE*, *83*(8).

MMOX. (2009). Wiki: Second life architecture working group. Retrieved August 10, 2009, from http://wiki.secondlife.com/wiki/MMOX

Multiverse. (2009). Multiverse platform architecture. Retrieved September 9, 2009, from http://update.multiverse.net.wiki/index.php/

Nae, V., Iosup, A., Podlipnig, S., Prodan, R., Epema, D., & Fahringer, T. (2008). Efficient management of data center resources for massively multiplayer online games. *Proceedings of the 2008 ACM/IEEE Conference on Supercomputing*. Austin, Texas.

Narayanasamy, V., Wong, K. W., Fung, C. C., & Rai, S. (2006). Distinguishing games and simulation games from simulators. *Computers in Entertainment*, *4*(2).

Naval Postgraduate School. (2009). Cluster & resources. Retrieved August 30, 2009, from http://www.nps.edu/Technology/HPC/ClusterResources.html

Noseworthy, J. R. (2008). The Test and Training Enabling Architecture (TENA) supporting the decentralized development of distributed applications and LVC simulations. *Proceedings of the 2008 12th IEEE/ACM international Symposium on Distributed Simulation and Real-Time Applications*. Washington, DC.

Office of Technology Assessment. U.S. Congress. (OTA). (1995). *Distributed interactive simulation of combat*, OTA-BP-ISS-151. Washington, DC: U.S. Government Printing Office.

Office of the Secretary of Defense. (2001). *Joint publication 1-02, department of defense dictionary of military and associated terms*. Retrieved August 12, 2009, from http://www.dtic.mil/doctrine/jel/doddict

OGC. (2008). Keyhole Markup Language (KML) open geospatial consortium standard. Retrieved August 22, 2009, from http://www.opengeospatial.org/standards/kml

Olson, M. A., Bostic, K., & Seltzer, M. (1999). Berkeley DB. *Proceedings of the FREENIX Track: 1999 USENIX Annual Technical Conference*. Monterey, California.

Page, E. H., Briggs R., & Tufarolo, J. A. (2004). Toward a family of maturity models for the simulation interconnection problem. *Proceedings IEEE Spring Simulation Interoperability Workshop*. IEEE CS Press.

Pantel, L., & Wolf, L. C. (2002). On the impact of delay on real-time multiplayer games. *Proceedings of the 12th international Workshop on Network and Operating Systems Support for Digital Audio and Video*. Miami, Florida.

Park, H., Kim, T., & Kim, S. (2005). Network traffic analysis and modeling for games. *Proceedings Workshop on Internet and Network Economics 2005*, LNCS 3828, 1056–1065.

Pelligrino, J., & Dovrolis, C. (2003). Bandwidth requirement and state consistency in three multiplayer game architectures. *Proceedings of Netgames '03*. Redwood City, California.

Portico Project. (2009). Open source, cross-platform HLA RTI implementation. Retrieved September 23, 2009, from http://porticoproject.org/index.php?title=Main_Page

Project Darkstar. (2009). Retrieved August 11, 2009, from http://projectdarkstar.com

Pullen, M. (2000). *Understanding internet protocols through hands-on programming.* New York: John Wiley & Sons.

Rauch, T. M. (2006). *Savage Modeling and Analysis Language (SMAL) metadata for tactical simulations and X3D visualizations*. Master's Thesis, Naval Postgraduate School, Monterey, California.

Ries, M., Svoboda, P., & Rupp, M. (2008). Empirical study of subjective quality for massive multiplayer games. *Proceedings of 15th International Conference on Systems, Signals and Image Processing, IWSSIP 2008*. Bratislava, Slovakia.

Rosedale, P., & Ondrejka, C. (2008). Enabling player-created worlds with grid computing and streaming. *Gamasutra*. Retrieved August 3, 2009, from http://www.gamasutra.com/resource_guide/20030916/rosedale_pfv.htm

Rushby, J. (1994). Critical system properties: Survey and taxonomy. *Reliability Engineering and System Safety, 43*(2), 189-219.

Ryan, P. Z., & Zalcman, L. (2003). The DIS vs. HLA debate: What's in it for Australia? *Proceedings of SimTect 2003.* Simulation Industry Association of Australia.

Sanders, K. (2007). *Requirements to create a persistent, open source, mirror world for military applications*. Master's Thesis, Naval Postgraduate School, Monterey, California.

Sauerborn, G. C. (1999). The distributed interactive simulation (DIS) lethality communication server, volume I: Overview. ARL-TR-1775.

Simulation Interoperability Standards Organization (SISO). (2007). Guide for: DIS plain and simple. SISO-REF-020-2007.

Singhal, S., & Zyda, M. (1999). *Networked virtual environments: design and implementation*. ACM Press/Addison-Wesley Publishing Co.

Smith, R. (2009, April). Games: The virtual frontier. *Presentation to Canadian Training Technologies Conference*. Retrieved September 4, 2009, from http://www.peostri.army.mil/CTO/

Sony Online Entertainment. Retrieved September 9, 2009, from http://www.soe.com/soe.vm

Steel, J. (2000). The use of DIS and HLA for real-time virtual simulation–A discussion. *Presentation at the Second NATO Modeling and Simulation Conference*.

Sterling, B. (1993). War is virtual hell. *Wired 1.01*.

Strassburger, S., Schulze, T., & Fujimoto, R. (2008). Future trends in distributed simulation and distributed virtual environments: Results of a peer study. *Proceedings of the 2008 Winter Simulation Conference*. IEEE, 777–785.

Tanenbaum, A. S., & Steen, M. V. (2002). *Distributed systems principles and paradigms*. New Jersey: Prentice Hall.

Tatham, S. PuTTY: A free telnet/SSH client. Retrieved July 10, 2009, from http://www.chiark.greenend.org.uk/~sgtatham/putty/

Tay, V. (2005). Massively multiplayer online game (MMOG)—A proposed approach for military application. *Proceedings of the 2005 International Conference on Cyberworlds*.

Technet. (2009). TCPNoDelay. Retrieved September 4, 2009, from http://technet.microsoft.com/en-us/library/cc783904(WS.10).aspx

Tolk, A., & Muguira, J. A. (2003). The levels of conceptual interoperability model. *Simulator Interoperability Working Group (SISO) Fall Workshop, 2003*.

Tolk, A., & Turnitsa, C. D. (2007). Conceptual modeling of information exchange requirements based on ontological means. *Proceedings of the 39th Conference on Winter Simulation: 40 Years! The Best is yet to Come*. Washington, DC.

Twesme, J., & Corzine, A. (2003). Naval Air Systems Command (NAVAIR) Unmanned Aerial Vehicle (UAV)/Unmanned Combat Aerial Vehicle (UCAV) distributed simulation infrastructure. *Proceedings of the 2nd AIAA "Unmanned Unlimited" Conference*, San Diego, California.

U. S. Department of Defense. Under Secretary of Defense for Acquisition and Technology. (2000). Memorandum, Subj: DoD High Level Architecture (HLA) for Simulations Memorandum of Agreement. Retrieved June 10, 2009, from http://acquisition.navy.mil/content/view/full/965

U.S. Army Virtual Targets Center. (2009). Army Model Exchange. Retrieved September 20, 2009, from https://modelexchange.army.mil

U.S. Congress, Office of Technology Assessment. (1995). *Distributed interactive simulation of combat*, OTA-BP-ISS-151. Washington, DC: U.S. Government Printing Office.

USAF Aeronautical Systems Center. (1998). Air force distributed mission training technical analysis. *Training Systems Program Office*, Wright Patterson Air Force Base.

Web3D Consortium. (2009). X3D Resources. Retrieved September 23, 2009, from http://www.web3d.org/x3d/content/examples/X3dResources.html

Weekley, J., Brutzman, D., Healey, A., Davis, D., & Lee, D. (2004). AUV workbench: integrated 3D for interoperable mission rehearsal, reality and replay. *Proceedings of 2004 Mine Countermeasures and Deming Conference*. Canberra, Australia.

Wireshark. (2009). Retrieved July 12, 2009, from http://www.wireshark.org/

Woodcock, B. S. (2008). An analysis of MMOG subscription growth. Presentation to ION 08. Retrieved June 4, 2009, from http://www.mmogchart.com

X-Plane SDK. (2009). Retrieved July 30, 2009, from http://www.xsquawkbox.net/xpsdk/mediawiki/Main_Page

Yap, K. M., Marshall, A., Yu, W., Dodds, G., Gu, Q., & Souayed, R. (2005). Characterising distributed Haptic virtual environment network traffic flows. *Proceeding of IFIP Network Control and Engineering for QoS, Security and Mobility*. France.

Yasui, T., Ishibashi, Y., & Ikedo, T. (2005). Influences of network latency and packet loss on consistency in networked racing games. *Proceedings of Netgames '05*, Hawthorne, New York.

Ye, M., & Cheng, L. (2006). System-performance modeling for massively multiplayer online role-playing games. *IBM Systems Journal*, *45*(1), 45–58.

Zyda, M. (2005). Interoperability & games. *Presentation Given at the Fall 2005 Simulation Interoperability Workshop*. Orlando, Florida. Retrieved July 10, 2009, from http://gamepipe.usc.edu/~zyda/Presentations.html

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California

3. Don Brutzman
   Naval Postgraduate School
   Monterey, California

4. Don McGregor
   Naval Postgraduate School
   Monterey, California

5. Amela Sadagic
   Naval Postgraduate School
   Monterey, California

6. Bob Deforest
   Air Force Agency for Modeling & Simulation
   Orlando, Florida

7. Garth Jensen
   Office of Naval Research
   Arlington, Virginia

8. Doug Maxwell
   Naval Undersea Warfare Center
   Newport, Rhode Island

9. John Moore
   Navy Modeling & Simulation Office
   Washington, D.C.

10. Robert Marble
    United States Special Operations Command Headquarters
    MacDill AFB, Florida

11. Maj James Neushul, USMC
I MEF Future Ops
Camp Pendleton, California

12. Loren Peitso
Naval Postgraduate School
Monterey, California