



Calhoun: The NPS Institutional Archive

DSpace Repository

Theses and Dissertations | Thesis and Dissertation Collection

2000-09

3D visualization of Theater-level Radio Communications using a networked virtual environment

Laflam, David W.

Monterey, California. Naval Postgraduate School

http://hdl.handle.net/10945/42175

Downloaded from NPS Archive: Calhoun



http://www.nps.edu/library

Calhoun is a project of the Dudley Knox Library at NPS, furthering the precepts and goals of open government and government transparency. All information contained herein has been approved for release by the NPS Public Affairs Officer.

Dudley Knox Library / Naval Postgraduate School
411 Dyer Road / 1 University Circle
Monterey, California USA 93943

NAVAL POSTGRADUATE SCHOOL
Monterey, California



THESIS

3D VISUALIZATION OF THEATER-LEVEL RADIO COMMUNICATIONS USING A NETWORKED VIRTUAL ENVIRONMENT

by

David W. Laflam

September 2000

Thesis Advisor:	Don Brutzman
Thesis Co-Advisor:	Michael Capps
Second Reader:	Don McGregor

Approved for public release; distribution is unlimited.

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	3. REPORT TYPE AND DATES COVERED
	September 2000	Master's Thesis

4. TITLE AND SUBTITLE
3D Visualization of Theater-Level Radio Communications Using a Networked Virtual Environment.

5. FUNDING NUMBERS

6. AUTHOR
David W. Laflam

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Naval Postgraduate School
Monterey, CA 93943-5000

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

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

The military is heavily reliant on the transfer of information among various networks in day-to-day operations. Radio-based communications networks that support this volume of information are complex, difficult to manage, and change frequently. Communications network planners need a way to clearly visualize and communicate mobile operational network capabilities, particularly to network users.

By using the DIS-Java-VRML simulation and modeling toolkit, visualizations of radio-frequency energy and radio path-profiling data can be quickly generated as 3D models. These animated 3D visualizations can be loaded into a networked virtual environment, so that communications planners can detect a variety of problems such as radio frequency interference and gaps in coverage. Planners can also brief senior staff, plan within their own staff, and collaborate with communications staff planners in distant locations using such virtual environments.

DIS-Java-VRML visualization tools can provide a clear picture of the battle space with respect to the deployed communications architecture. The prototypes presented in this thesis demonstrate the ability to generate a shared visualization that can show a radio communications network in 3D. Such dynamic visualizations increase communications planning information bandwidth and yield more intuitive ways of presenting information to users. Higher information density in a more intuitive format enables better understanding with quicker reaction times. This thesis and the visualization tool discussed provide the foundation for fundamental improvements in visualizing radio communications environments.

14. SUBJECT TERMS	15. NUMBER OF PAGES
Virtual Environments, Visual Simulation, Signal Planning, VRML, Java, DIS-Java-VRML, X3D	
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release; distribution is unlimited

3D VISUALIZATION OF THEATER-LEVEL RADIO COMMUNICATIONS USING A NETWORKED VIRTUAL ENVIRONMENT

David W. Laflam
Captain, United States Army
B.S., Keene State College, New Hampshire 1989

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN MODELING VIRTUAL ENVIRONMENTS AND SIMULATION

from the

NAVAL POSTGRADUATE SCHOOL
September 2000

Author: _____

David W. Laflam

Approved by: _____

Don Brutzman, Thesis Advisor

_____

Michael Capps, Thesis Co-Advisor

_____

Don McGregor, Second Reader

_____

Rudy Darken, Academic Associate
Modeling Virtual Environments and Simulation Academic Group

_____

Michael Zyda, Chair
Modeling Virtual Environments and Simulation Academic Group

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

The military is heavily reliant on the transfer of information among various networks in day-to-day operations. Radio-based communications networks that support this volume of information are complex, difficult to manage, and change frequently. Communications network planners need a way to clearly visualize and communicate mobile operational network capabilities, particularly to network users.

By using the DIS-Java-VRML simulation and modeling toolkit, visualizations of radio-frequency energy and radio path-profiling data can be quickly generated as 3D models. These animated 3D visualizations can be loaded into a networked virtual environment, so that communications planners can detect a variety of problems such as radio frequency interference and gaps in coverage. Planners can also brief senior staff, plan within their own staff, and collaborate with communications staff planners in distant locations using such virtual environments.

DIS-Java-VRML visualization tools can provide a clear picture of the battle space with respect to the deployed communications architecture. The prototypes presented in this thesis demonstrate the ability to generate a shared visualization that can show a radio communications network in 3D. Such dynamic visualizations increase communications planning information bandwidth and yield more intuitive ways of presenting information to users. Higher information density in a more intuitive format enables better understanding with quicker reaction times. This thesis and the visualization tool discussed provide the foundation for fundamental improvements in visualizing radio communications environments.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

I. INTRODUCTION 1
A. THESIS STATEMENT 1
B. MOTIVATION 1
C. OBJECTIVES 4
D. METHODOLOGY 4
E. ORGANIZATION OF THESIS 5
II. BACKGROUND AND RELATED WORK 7
A. INTRODUCTION 7
B. MILITARY AND CIVILIAN SECTORS RELIANCE ON NETWORKS 7
C. RADIO PROFILING 8
D. IEEE DISTRIBUTED INTERACTIVE SIMULATION PROTOCOL (DIS) 12
E. DISTRIBUTED INTERACTIVE SIMULATION CATAGORIES 13
1. PDU Descriptions 13
F. MULTICAST 14
G. JAVA 15
H. VIRTUAL REALITY MODELING LANGUAGE (VRML) 16
I. DIS-JAVA-VRML 31
J. EXTENSIBLE 3D (X3D) 32
K. TOOL REQUIREMENTS 37
L. COMMUNICATIONS VISUALIZATION TOOLS 37
1. OPNET Modeler™ 37
2. COMNET/STK 39
3. MSE-NPT 42
M. SUMMARY 44
III. METHODOLOGY 45
A. INTRODUCTION 45
B. OPPORTUNITY STATEMENT 45
C. PROPOSED IMPLEMENTATION 46
D. TECHNOLOGICAL SOLUTIONS 49
1. DIS DESIGN 49
2. EXTENSIBLE 3D (X3D) DESIGN 49
3. JAVA DESIGN 50
E. SUMMARY 51
IV. DIS EMISSIONS AND PDUS 53
A. INTRODUCTION 53
B. IEEE DIS DATA STREAM PDUs 53
C. DIS IMPLEMENTATION 53
D. RADIO COMMUNICATIONS PROTOCOL (RCP) FAMILY 57
1. Transmitter PDU 58
2. Receiver PDU 62
3. Signal PDU 65
E. SUMMARY 68
V. TACTICAL VISUALIZATION OF BATTLEFIELD EMISSIONS 69
A. INTRODUCTION 69
B. OVERVIEW 69
C. VISUALIZATION CONSIDERATIONS 70
1. Tactical Visualization 70
2. Dimensional Space 73
3. Available Graphics Techniques and Parameters 74
4. Mapping MSE-NPT Data to Graphics Parameters 75
5. Initial Visualization Recommendations 76
D. SUMMARY 78
VI. SYSTEMS INTEGRATION AND DEMONSTRATION 79
A. INTRODUCTION 79

B. DEMONSTRATION 79
C. VISUALIZATION RESULTS 90
D. PROPOSED USES OF RADIO COMMUNICATIONS VISUALIZATION 93
E. SCIENTIFIC VISUALIZATION AND INTEROPERABILITY 94
F. SUMMARY 94
VII. CONCLUSIONS AND RECOMMENDATIONS 95
A. INTRODUCTION 95
B. PRINCIPAL THESIS CONCLUSIONS 95
C. SPECIFIC CONCLUSIONS 95
1. Tactical Communications Visualization 95
2. Visualization 96
D. RECOMMENDATIONS FOR FURTHER WORK 96
1. Tactical Communications Visualization 96
2. Model Validation 97
3. Visualization 97
APPENDIX A. MSE-NPT File Reader Code 99
APPENDIX B. DIS PDU CODE 111
APPENDIX C. DIS-Java-VRML DTD 129
APPENDIX D. X3D PROTOS 133
APPENDIX E. CD-ROM 137
LIST OF REFERENCES 147
INITIAL DISTRIBUTION LIST 151

LIST OF FIGURES

Figure 2.1: Allocation of Radio Spectrum in the United States. *8*
Figure 2.2: Frequency Ranges of Radio Spectrum and uses in the United States. *9*
Figure 2.3: Fresnel Zones of an antenna array, showing optimal (1) to marginal (3) Zones of signal power, From Ref (TB 11-5895-1544-10-1). *10*
Figure 2.4: Reflection and refraction of radio waves From Ref (TB 11-5895-1544-10-1). *11*
Figure 2.5: "Hello World" Source Code and Rendered World From (Brutzman, 1998). *17*
Figure 2.6: SHF MSE Antenna in VRML World (Cylinder and Cone Nodes). *19*
Figure 2.7: 2Km by 2Km VRML IndexedFaceSet of Fort Irwin California generated From NIMA DTED Data. *20*
Figure 2.8: MSE UHF antenna using Transform Grouping of Cylinder Cone and IndexedFaceSet. *22*
Figure 2.9: View Point High Above Airfield with communication coverage area shown as green half domes. *23*
Figure 2.10: View Point close in next to Blue Flag with communication coverage. *23*
Figure 2.11: "Hello World" Source Code and Rendered World redone in GEOVrml. *29*
Figure 2.12: Generated VRML terrain of Valley in Cairngorms (vertically exaggerated) with an overlay of 1:25000 Map which was in GIF format Form Ref (McCullagh 1997). *30*
Figure 2.13: Screen capture of X3D-Edit Tool with allowed node-menu. *34*
Figure 2.14: Screen Capture of X3D-Edit Tool showing SHF Antenna Prototype. *35*
Figure 2.15: X3D Source Code for "Hello World". *36*
Figure 2.16: VRML Source Code for "Hello World". *36*
Figure 2.17: Satellite Tool Kit (STK) screen snapshot of 2D radio profiling. *41*
Figure 2.18: Satellite Tool Kit (STK) screen snapshot of 3D satellite profiling. *41*
Figure 3.1: Methodology for Signal Planning Visualization. *46*
Figure 3.2: Implementation methodology overview for signals visualization. *47*
Figure 3.3: Detailed implementation methodology for signal visualization. *48*
Figure3.4: Detailed Java connectivity design reads value from the network and sends data to and from the VRML scene. *51*
Figure 4.1: Instantiation of a Multicast socket using Java's built-in networking. *54*
Figure 4.2: DIS networking paths to create entities in the visualization. *55*
Figure 4.3: DIS-Java-VRML streaming stack, as documented in Javadoc from the mil.navy,nps.dis consistent initialization EntityDispatcher *package.* *56*
Figure 4.4: Text output of the start of the Antenna Visualization sending PDUs. *57*
Figure 4.5: Transmitter PDU Diagram From Ref (DIS IEEE Std 1278.1a-1995). *60*
Figure 4.6: Text output of test program mil.navy.nps.testing.TransmitterPduSender showing sending of a Transmitter PDU. *60*
Figure 4.7: Text output of test program mil.navy.nps.testing.PduTestListener showing proper receipt of a Transmitter PDU. *61*
Figure 4.8: Receiver PDU Diagram From Ref (DIS IEEE Std 1278.1a-1995). *63*

Figure 4.9: Text output of test program mil.navy.nps.testing.ReceiverPduSender showing sending of a Receiver PDU. 63
Figure 4.10: Text output of test program mil.navy.nps.testing.PduTestListener showing proper receipt of a Receiver PDU. 64
Figure 4.11: Signal PDU Diagram From Ref (DIS IEEE Std 1278.1a-1995). 66
Figure 4.12: Text output of test program mil.navy.nps.testing.SignalPduSender showing sending of a Signal PDU. 66
Figure 4.13: Text output of test program mil.navy.nps.testing.PduTestListener showing proper receipt of a Signal PDU. 67
Figure 5.1: MSE-NPT Terrain Profile From Ref (TB 11-5895-1544-10-1). 71
Figure 5.2: MSE-NPT Terrain Screen Shot of DTED Map From Ref (TB 11-5895-1544-10-1). 72
Figure 5.3: 7 major tasks for design of information visualization (Keller 1993). 74
Figure 5.4: MSE-NPT antenna data for coordinates, height and elevation are mapped directly to 3D rendering coordinates. 76
Figure 5.5: MSE-NPT file with VRML coordinates converted from MRGS coordinates. 77
Figure 6.1: DIS-Java-VRML desktop Icons. 79
Figure 6.2: DIS-Java-VRML Start Panel for Capture the Flag entities. 80
Figure 6.3: Text output of DIS-Java-VRML Referee copied from console. 81
Figure 6.4: Netscape Java Security warning for reading unsigned class files. 82
Figure 6.5: MSE-NPT loader selection panel. 83
Figure 6.6: MSE-NPT file loader dialog box. 83
Figure 6.7: MSE-NPT File with VRML Mission Coordinates. 84
Figure 6.8: Antenna Start Panel with selection choices generated by file parser. 85
Figure 6.9: MSE-NPT file data flow for ESPDU state information. 86
Figure 6.10: MSE-NPT File Data Flow for Radio Communications Family PDUs. 86
Figure 6.11: Antenna DIS-Java-VRML visualization startup sequence after the user has selected a startup MSE-NPT data file. 87
Figure 6.12: Antenna Control Panel allowing user to move, rotate, and send PDUs. 87
Figure 6.13: SignalPDU *Sent Across the DIS Network Before and After Visualizations...* 89
Figure 6.14: DIS-Java-VRML Antenna Signal Visualization far viewpoint. 90
Figure 6.15: DIS-Java-VRML Antenna Signal Visualization Close in Viewpoint showing 10 KM RAU Antenna Coverage. 91
Figure 6.16: DIS-Java-VRML antenna signal visualization with 10 kilometer overlapping coverage domes. 91
Figure 6.17: DIS-Java-VRML Antenna Signal Visualization with a viewpoint high above the Fort Irwin. 92
Figure 6.18: DIS-Java-VRML Antenna Signal Visualization with a viewpoint high above the airfield. 92
Figure 6.19: DIS-Java-VRML Antenna Signal Visualization with a viewpoint 20 Km above the airfield. 93

List of Acronyms

AOIM	-	Area Of Interest Management
ASCII	-	American Standard Code for Information Interchange
CAVE	-	CAVE Automatic Virtual Environment
CECOM	-	Communications Electronics Command (US Army)
DIS	-	IEEE Distributed Interactive Simulation Protocol
DoD	-	Department of Defense
ESPDU	-	Entity State PDU
GUI	-	Graphical User Interface
IEEE	-	Institute of Electrical and Electronics Engineers
ISO	-	International Standards Organization
JVM	-	Java Virtual Machine
LCD	-	Liquid Crystal Display
LOD	-	Level of Detail
LOS	-	Line Of Sight
LSVE	-	Large Scale Virtual Environment
MGRS	-	Military Grid Reference System
MSE-NPT	-	Mobile Subscribers Equipment Network Planning Tool
NIPRNet	-	Unclassified but Sensitive Internet Protocol Router Network
NPS	-	Naval Postgraduate School
NPT	-	Network Planning Terminal (See MSE-NPT)
PDU	-	Protocol Data Unit
SIGGRAPH	-	Special Interest Group for GRAPHics
VE	-	Virtual Environment
VRML	-	Virtual Reality Modeling Language
X3D	-	Extensible 3D
XML	-	Extensible Modeling Language

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGEMENTS

I would like to express my sincere appreciation to my classmates. Without their long hours at the library and in review sessions this degree would not have been possible. We were a good team. I would also like to thank my thesis advisors, Dr. Don Brutzman, Dr. Michael Capps and Don McGregor for their wise counsel, patience and dedication throughout my work on this thesis.

I would especially like to thank Dr. Michael Zyda, the creator of the Modeling, Simulations and Virtual Environments (MOVES) curriculum at the Naval Postgraduate School. Dr Zyda is an outstanding educator and a visionary in his field. He has provided the insight necessary to understand the complex problems facing the Army's modeling and simulation community while providing the analytic and cognitive tools necessary to find workable solutions for those problems.

Finally, I must recognize the loving support of my family, who supplied support through phone calls and email messages. To my father, Colonel Robert J. Laflam (Ret), who has always kept me focused on the bigger picture and to my mother, Margaret B. Laflam, for always being there when I needed a sounding board. Thanks for always being there.

I. INTRODUCTION

A. THESIS STATEMENT

The DIS-Java-VRML software toolkit can be used to dynamically visualize radio-wave propagation on realistically rendered three-dimensional (3D) terrain, thereby enhancing the capability of the Army's Mobile Subscriber Equipment Networking Planning Terminal (MSE-NPT).

B. MOTIVATION

The military is heavily reliant on the transfer of information among various networks in its day-to-day operations. With fewer defense dollars available for the development of new systems, the use of commercial-off-the-shelf (COTS) hardware and software to build military information networks is becoming commonplace. The critical nature of much of this radio profiling and planning information requires that knowledge regarding communications-network performance characteristics be well understood. These characteristics allow network managers and designers to plan for future growth of the network, analyze network reliability, and plan for the construction of new networks. This problem is often addressed satisfactorily for wired networks but is rarely addressed for mobile radio communications networks.

When presented with a problem containing dynamic sources of information, it is easiest to understand the overall impact of a solution if one addresses the problem in a visual manner. Because the world is inherently 3D, people often tend to visualize spatial

solutions to problems in 3D. Most people tend to think in 3D, yet most of the data they interact with on a daily basis is presented in a two-dimensional (2D) format. While many people have a high degree of spatial ability, none of this capability is exploited when they only process in 2D. Individuals rarely communicate verbal information in a 2D aspect. More often than not, people verbalize with respect to a 3D reference. This is scene when a person gives someone directions to a store. People tend to give directions in a 3D framework, such as "Go down the street and take a left," as opposed to, "Go up the map and across." This points to the fact that the human brain has limitations in its ability to process large volumes of numeric information, such as points on a map, to then make conclusions about likely outcomes. This leads one to believe that more innovative means of visualization are necessary to allow the signal planner and commander to step another level of abstraction away from raw profile data, to facilitate timely and accurate decision-making.

When military radio-communications planners can view planning-system inputs in a graphical context, it provides them the opportunity to rapidly recognize potential radio-profiling problem areas. Such profiling conflicts easily remain undetected when the same information is presented in a text-based form. In the communications field, systems that communicate must have a physical path from the sender or transmitter to the receiver. By viewing the radio-transmission communication paths visually in the context of geographic and tactical obstacles, the systems planner can quickly identify whether a planned system has problems. Currently such analysis (when performed at all) utilizes topographic maps and manually sketched overlay data.

This work assumes that if a communications planner can view radio communications systems in a 3D space, the problems between the sender and receiver can be more quickly identified, as will integration problems among all the systems in a network. Such visual detection of problem areas is an active field of study, as described in "Visual Discovery and Analysis" (Eick 2000). By providing a visualization enhancement to currently fielded systems, this work can improve a communications staff's ability to support warfighting commanders. The system presented in this thesis enables them to produce and create a 3D signal flow in the designated battle space.

Performance characteristics of a network can be determined through modeling and simulation. The DIS-Java-VRML modeling software toolkit which supports IEEE and ISO standards can be used as an extension to traditional COTS radio-modeling software to provide 3D visualization of the radio propagation patterns. Producing simulations and models that are based on ISO and IEEE standards ensures that the developer of the simulation enjoys the highest degree of integration with existing simulations. This type of solution also provides the greatest amount of flexibility when contracting for additional software support, by allowing the greatest number of developers to bid on the development of the software. Standards also enable local development of additional capabilities. This thesis demonstrates how to capture information from an ASCII text-based form and transform it into a 3D scene for the modeling and simulations of radio-based networks.

C. OBJECTIVES

The objective of this thesis is to demonstrate the viability of using Extensible Markup Language (XML)/ Extensible 3D (X3D) and Virtual Reality Modeling Language (VRML) to render an ASCII MSE-NPT file into a 3D tactical battlefield visualization system to aid radio-network planners. To develop a tactical battlefield visualization system, the following essential software components must be developed:

- Tactical battlefield visualization system environment designed in VRML
- An interface that will transform relevant NPT data from ASCII text into appropriate VRML and X3D (Extensible 3D) objects.
- An interface that will allow the user to reposition any of the antennas described in the NPT File on realistically rendered terrain
- Java based DIS Software for transmitter, receiver and signals
- VRML PROTO libraries that are referenced by the DIS code

D. METHODOLOGY

The following steps were taken in order to address the above issues:

Background Study: The background notes outline all major radio communications software currently used within the Department of Defense (DoD). It also explains the capabilities these software packages have, focusing on their ability to visualize information in 3D as well as how these packages share data across a network.

Framework Development: Develop a battlefield visualization framework based on the DIS-Java-VRML toolkit. This visualization is developed to demonstrate reading in data from the network and generating 3D visualization of radio communications on a realistically rendered terrain model.

Demonstration: The viability of this framework is demonstrated on a prepared set of MSE-NPT Data. Information can be represented in both the current form of VRML and also natively encoded into the coming generation of VRML called Extensible 3D (X3D).

E. ORGANIZATION OF THESIS

The remainder of this thesis is organized as follows:

- **Chapter I: Introduction.** This chapter includes an introduction to the problem, and the steps in its solution.
- **Chapter II: Background.** This chapter provides the background and related work on the reason for visualizing communications, introducing some of the tools currently used by the radio network design community.
- **Chapter III: Methodology.** This chapter presents the problem statement and covers design considerations in enhancing the Army's MSE-NPT visualization capabilities with a DIS-Java-VRML solution.
- **Chapter IV: DIS Emissions and PDUs.** This chapter presents the development of the DIS compliant networking code for radio communications, and explains why the DIS Standard is a good choice for large-scale virtual environments (LSVEs).
- **Chapter V: Tactical Visualization of Battlefield Emissions.** This chapter introduces how tactical visualization of battlefield emissions are currently done, and how use of the DIS-Java-VRML solution can improve visualization.

- **Chapter VI: Systems Integration and Demonstration.** This chapter provides a description of the DIS-Java-VRML framework, and demonstrates its implementation, to include how the framework can read in an external MSE-NPT file and generate a 3D scene. Example visualizations results are presented in detail.
- **Chapter VIII: Conclusions and Recommendations.** This chapter contains the conclusion reached and recommendations for future development with DIS-Java-VRML in the problem area.

II. BACKGROUND AND RELATED WORK

A. INTRODUCTION

This chapter reviews the fundamental concepts which are the basis for the development and generation of 3D radio-communications profile plans. Topics examined in this chapter include the military and civilian-sector reliance on radio-based communications networks, radio profiling, Distributed Interactive Simulation (DIS), Java, Virtual Reality Modeling Language (VRML), DIS-Java-VRML and Extensible 3D (X3D), plus software-tool requirements and communications-visualization tools.

B. MILITARY AND CIVILIAN SECTORS RELIANCE ON NETWORKS

The trend in both civilian and military sectors has been toward heavy reliance on information networks in day-to-day operations. Rapid advances in information technology have propelled this shift from an industry-based economy to an information-centric economy. As a result, society has shifted away from production-based transport systems to information systems where the ability to transport information to the necessary destinations at the needed time and in a format which can quickly be summarized by either man or machine is the key to both corporate and military success.

This shift is particularly noticeable in the military. The military paradigm is rapidly evolving from Carl von Clausewitz's theory of massing combat power and overwhelming the enemy to that of the global infosphere, where data concerning the enemy may be transferred to the necessary location, delivering a precision strike without requiring the classical massing of forces. The dependence on networks within the

military might best be described by General Colin L. Powell's June 1992 assessment in of the use of networks during the Persian Gulf War:

> "Efficient management of the information increased the pace of combat operations, improved decision-making, and synchronized various combat capabilities. The technology developed to support these networks proved to be vital margin that saved lives and helped achieve victory." (Powell, 1995)

A variety of network technologies including IP, X.25 and ATM are connected over wireless networks that must transport a great deal of information. These wireless links must be engineered to support a force that is mobile, and mobility is achieved with radio systems. The best way to ensure the highest degree of transmission reliability between these radio systems is to visually profile the links between the radios. Radio profiling ensures the best communications throughput for the networks that are being supported by the radio transmission systems.

C. RADIO PROFILING

Radio profiling is one of the oldest and most common techniques used to ensure reliable communications between radio wave-based systems. The military has systems that use all bands of the radio spectrum. See Figure 2.1 and Figure 2.2 for a detailed description of available equipment by frequency range.

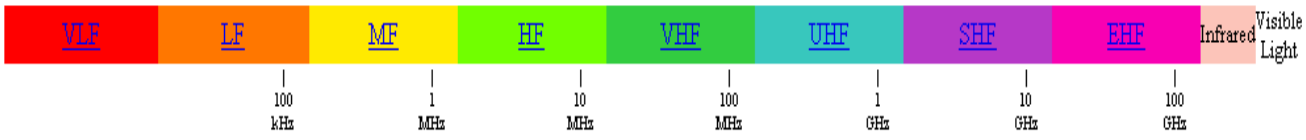


Figure 2.1: The allocation of radio spectrum in the United States.

0 kHz to 30 kHz	Very Low Frequency (VLF)	Maritime mobile telephone service
30 kHz to 300 kHz	Low Frequency (LF)	Radio beacons for aircraft navigation
300 kHz to 3 MHz	Medium Frequency (MF)	
3 MHz to 30 MHz	High Frequency (HF)	Army squad radios
30 MHz to 328.6 MHz	Very High Frequency (VHF)	
328.6 MHz to 450 MHz		
450 MHz to 470 MHz		Trunked or Conventional Base Radio
470 MHz to 806 MHz	Ultra High Frequency (UHF)	
806 MHz to 960 MHz		Greatest usage between 806 MHz to 960 MHz
960 MHz to 2.3 GHz		
2.3 GHz to 2.9 GHz		Wireless Communications Service
2.9 GHz to 30 GHz	Super High Frequency (SHF)	Transmission type for high capacity network systems
30 GHz and above	Extremely High Frequency (EHF)	Point-to-Point Microwave Service

Figure 2.2: Frequency ranges of radio spectrum and uses in the United States.

This thesis is focused on the frequency bands used by the US Army's Mobile Subscriber Equipment, which are the UHF and SHF radio bands.

Radio profiling is mapping the side or sectional elevation of the radio transmission path as it relates to the ground from the transmitter to the receiver. It is a graph that represents the extent to which an individual radio system exhibits traits as determined by its antenna wave emission contour shown as a drawing of vertical lines. The goal of the radio network engineer is to have a system with a minimum of obstacles in the radio path. Most radio antenna planners like to place their antennas on highest point above the ground, thus allowing for the greatest amount of line of sight (LOS)

between the emitters using the most direct path or zone. Zone 1 is the optimal zone, with the best LOS and a very direct path. Zone 3 is the least favored zone, with marginal LOS and a very indirect path.

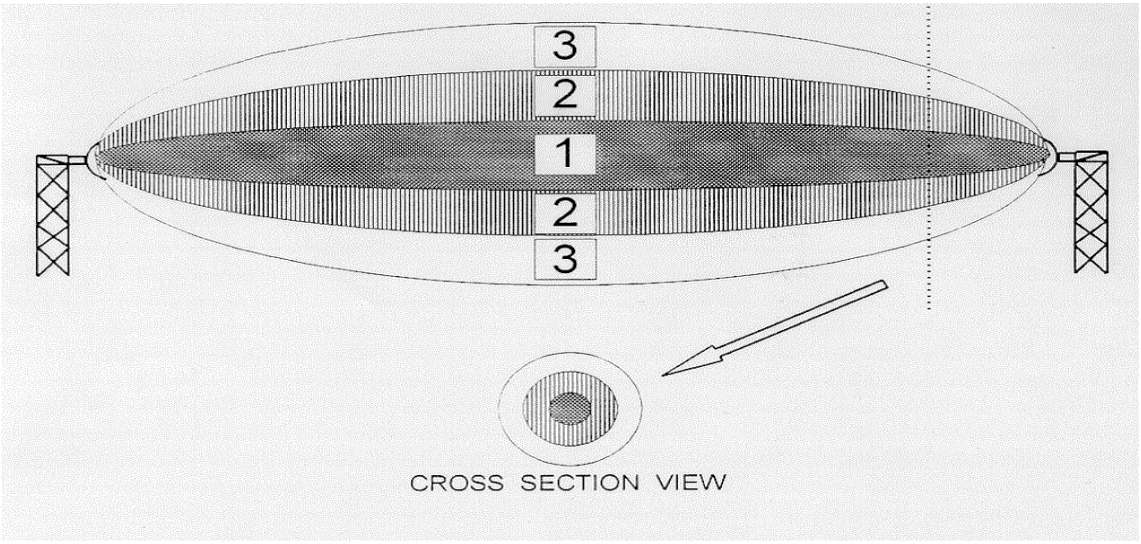


Figure 2-3: Fresnel Zones of an antenna array, showing Optimal (1) to Marginal (3) Zones of signal power, From Ref (TB 11-5895-1544-10-1).

LOS enables operators to quickly and easily determine the visible and non-visible regions around an antenna as viewed from an observer's position. A radio-path profile along the direct line of sight from the transmitter to the receiver is generated and shows visible or non-visible areas, based on the terrain elevation data and obstacles noted in the data set.

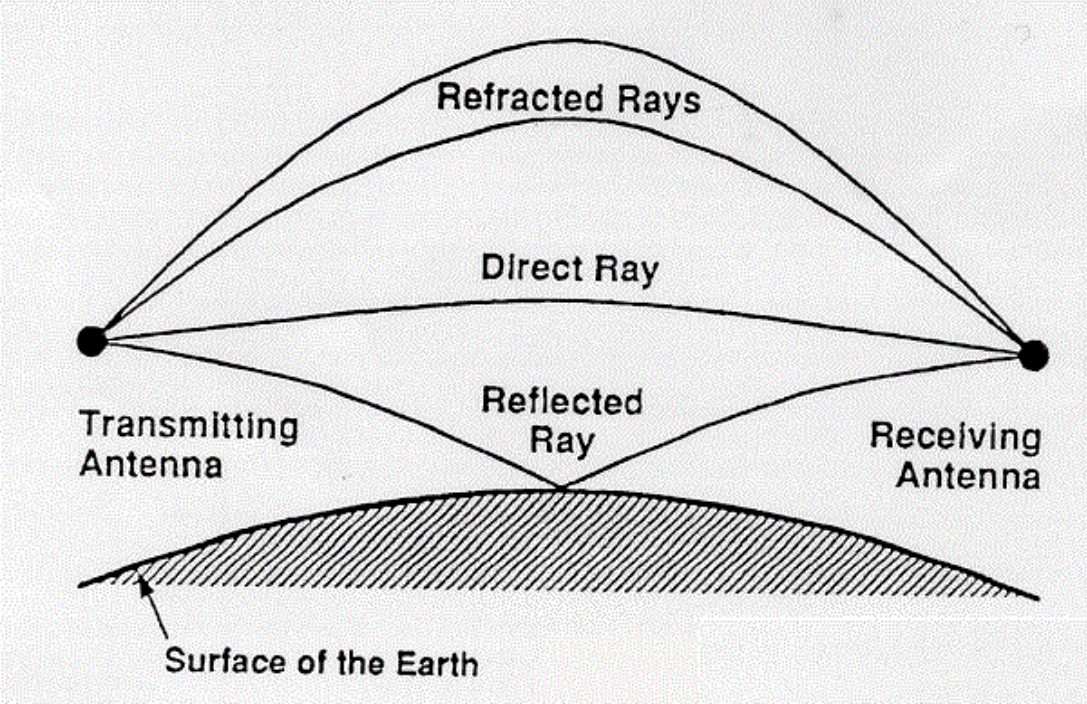


Figure 2-4: Reflection and refraction of radio waves
From Ref (TB 11-5895-1544-10-1).

Radio profiling is important to ensure that geographically dislocated areas, which have networks that must share data, can communicate to each other. Properly radio-profiling links between networks ensures that communication between commanders and their units is continuous regardless of where in the battle space they are physically located. This is also the trend in the commercial sector. Currently such efforts can be seen in the wide deployment of the Institute of Electrical and Electronics Engineers (IEEE) 802.11 compliant wireless LANs, where by various companies use wireless network connections to track shop floor inventory, to communicate with a sales staff, and even to assist with patient monitoring when transporting a person in an ambulance. By properly profiling the terrain in conjunction with antenna emplacement, the radio-signals

planner can have the greatest area of radio coverage. Such coverage allows the radio planner more flexible usage in that area of coverage. In the military, the greatest area of communications coverage translates into the greatest area of command and control.

D. IEEE DISTRIBUTED INTERACTIVE SIMULATION PROTOCOL (DIS)

The IEEE Standard for Distributed Interactive Simulation Application Protocols (DIS) is a government and industry initiative to define an infrastructure for linking simulations. The DIS protocol IEEE 1278.1 is a set of over-the-wire communications protocols used to standardize the way information about a simulation is packetized and transmitted "over the wire" (i.e. via a network connection) to other computers and computer networks. DIS was an outgrowth of the simulator-networking project (SIMNET) done by Bolt, Beranek and Newman for Defense Advanced Research Projects Agency (DARPA) in the late 1980's. The intent of this project was to link a number of computers together to create a virtual battlefield. DIS combines interactivity and distributed processing. The receiving computers can differ from the sending computer in architecture and operating systems. As long as all the computers in the system have implemented the DIS protocol correctly, they are able to communicate within a simulation. DIS messages are composed of a series of ordered data fields to ensure wide-scale interoperability, communicating state information (such as position, orientation, velocity and acceleration) among multiple entities participating in a shared network environment. The underlying transport protocol for the DIS packets is usually multicast.

E. DISTRIBUTED INTERACTIVE SIMULATION CATAGORIES

This protocol is broken down into five general categories for which there are PDUs. Each area provides a common description or specification for writing code, which has been agreed upon by the IEEE specification committee.

1. PDU Descriptions

a) **Entity State** –Provides information including appearance, location, velocity, orientation, acceleration, position and movement of articulated parts for simulated entities. Location and movement changes are dead reckoned and this PDU is sent at a variable rate necessary to correct dead reckoned parameters. The Entity State PDU may also be sent periodically as a heartbeat, or to compensate for lost PDUs.

b) **Emissions** – Provides and includes PDUs that control the point of origin, power, frequency, direction, scan pattern, and other parameters of electronic or acoustic emissions. This information is used to stimulate simulated sensors capable of detecting and responding to such information. In electronic warfare (EW) environments, these parameters change frequently. In an active EW environment, emission packets are sent as frequently as entity state packets.

c) **Data Stream PDUs** -- Represent voice samples, computer-to-computer communications, images, or any other digital bit stream. These are heavily used in simulations that include voice radio, intelligence and tactical command-and-control systems.

d) **Environment** – Breaks up atmospheric or oceanographic data into a series of PDUs, each of which describe changes in the simulated natural environment.

e) **Fire & Detonation** -- Carry the information needed to describe the firing of a ballistic (unguided) weapon and the detonation of the projectile. The amount of information per PDU is fairly small and the total series of PDUS are limited to the amount of ammunition the participants can carry. Weapon firing tends to come in bursts.

Interoperability is vital within the communications arena as well as within the modeling and simulation community. When a military command exchanges messages with a subordinate military command, each message needs to correctly received. Similarly, if a commander can share an identical picture of the battle space with subordinate commanders, the mission plan is more clearly understood. By being able to better visualize the radio network, planning officers can more accurately communicate problems relating to the radio communications infrastructure to a non-communications planning officer. Utilizing the DIS protocol with 3D virtual environments also provides an opportunity to share one's visualization with staff members located at distant locations.

F. MULTICAST

Multicast combines the best parts of two types of network communications: broadcast and point-to-point links. Any given host, in addition to having its own Internet address, can belong to a large number of multicast groups. Each multicast group has its

own special Internet address with in a certain range of addresses set aside for this purpose. When any host sends a message to a multicast address, that message is sent to the entire host range belonging to the multicast group. In effect, this approach is much like having a separate subnet on the Internet. Sending a multicast message is easier and more efficient than sending a copy of a point-to-point message to every participant on the network. Hosts that are not subscribed to the group can ignore the packets on the network at the hardware level. Since there is no central server and no duplicated bandwidth, multicast scales well. The lack of a central servers means that people can join and leave the network-simulation without having to worry about authentication procedure each time. The signals visualization presented in this thesis implements a DIS multicast between each network simulation.

G. JAVA

Java™ is a programming language specifically designed for use in the distributed environment of networks and Internet. Java can be used to create complete applications that may run on a single computer or be distributed among servers and clients in a network. The primary resource for this language is http://java.sun.com . Some of the noteworthy characteristics of Java include:

- The programs created are portable in a network. The program is compiled into Java byte code which can be run anywhere in a network on a server or client that has a Java virtual machine.
- The code is "robust," meaning that the Java objects can contain no references to data external to themselves or other known objects.

- Java is object-oriented, which means that, among other characteristics, similar objects can take advantage of being part of the same class and inherit common code.
  (Deitel, 1999)

Using Java as the development programming language insures that the system will be compatible with the greatest number of computer platforms.

H. VIRTUAL REALITY MODELING LANGUAGE (VRML)

Virtual Reality Modeling Language (VRML) is a language for describing Web-based 3D models and possible user interactions with them. One key feature of VRML is that it is an ISO standard designed to be used over the in a Web browser environment. Using VRML, a developer can integrate a set of visual models into a Web-based setting with which a user can experience by viewing, moving, rotating, and otherwise interacting with the scene. For example, a user can view a room and use controls within the web browser to move about the room, mimicking what they might see if one were walking through the room in real space. The various examples explored in this thesis employ the approved (VRML97) standard.

The fundamental design structure in VRML97 worlds is a scene graph. A 3D scene graph describes the three-dimensional world, including the objects it contains, their visual properties, their behaviors and how they interact. By encoding content into groups of nodes one can divide the scene into smaller parts, optimizing the scene. To display objects like primitive shapes such as Box, Sphere, Elevation grids, and complex Indexed Face Sets, one creates these nodes and adds to their behaviors. The nodes also specify groupings of sub-nodes and can indicate interaction and movement of events throughout

the scene graph. There are the two basic steps used to design a scene graph: building a world with visual nodes, and then describing the interaction through behavior nodes. VRML provides the standardized interchange language to create such virtual worlds so they can be viewed with any VRML-capable Web browser. Figure 2.5 implements several of these VRML nodes to create a virtual earth (Brutzman, 1998).

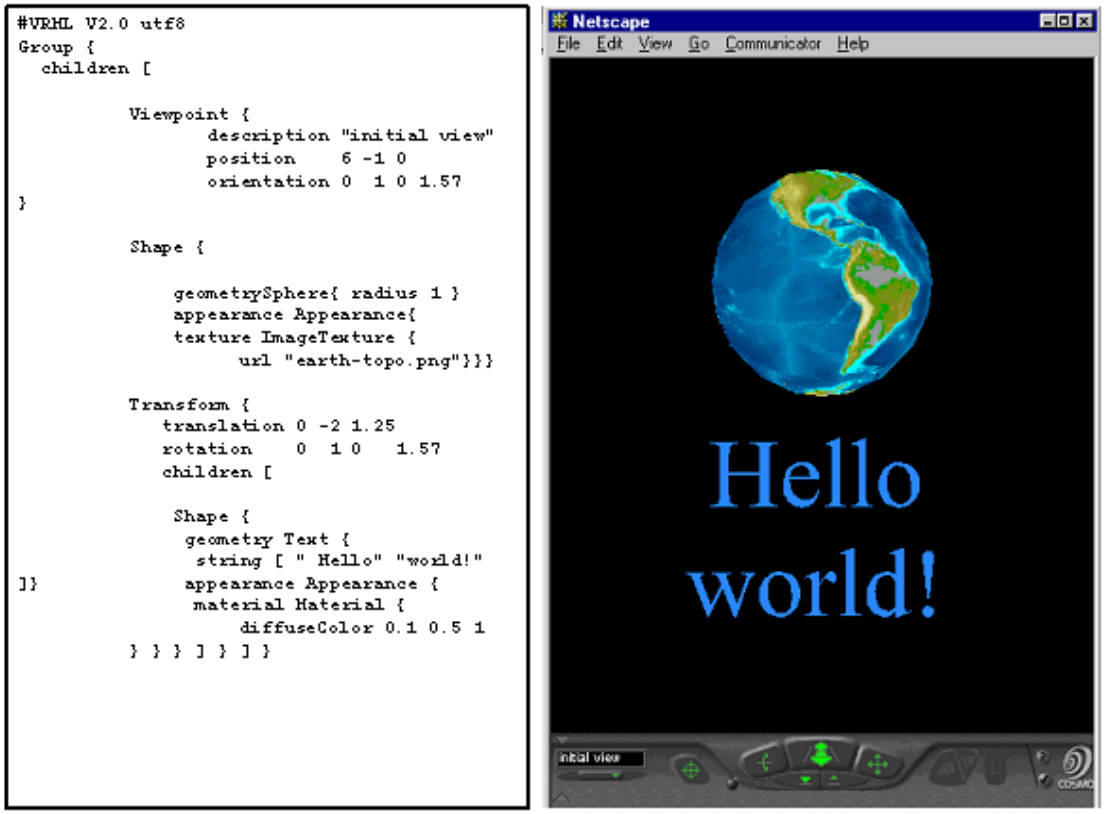


Figure 2.5: "Hello World" Source Code and Rendered World.
From Ref (Brutzman, 1998).

1. Basic Nodes

VRML has a rich variety of nodes that can be employed to develop a scene graph. The following is an overview of key nodes necessary to understand the design of the example communications visualization virtual world presented in this thesis.

a. Visual Nodes

The visual aspect of a scene graph is expressed through its geometry. In VRML the *Shape* node has four primitives: the Box, Cone, Cylinder and the Sphere. These primitive shapes are inserted into the scene graph as nodes and define a specific shape in the geometric field. These nodes can be grouped, sized, scaled, colored, and textured as appropriate to build entities in the virtual world.

The *Shape* node is also used to build more complex objects such as the MSE SHF antenna in Figure 2.6. These complex objects can be built using a combination of a cone and a cylinder or in the case of the terrain for the model it was generated as a VRML *IndexedFaceSet*. An *IndexedFaceSet* node is an array of polygons used to map out an object in the virtual world

In general, an object is often built with the help of a CAD (computer aided design) program or similar authoring tool. The software then converts and exports the object to a VRML compatible *IndexedFaceSet* node. For example, the terrain map in Figure 2.7 was built using a Java3D program the reads and generates VRML terrain files. It was written in cooperation with Mr. John Kim of NIMA.

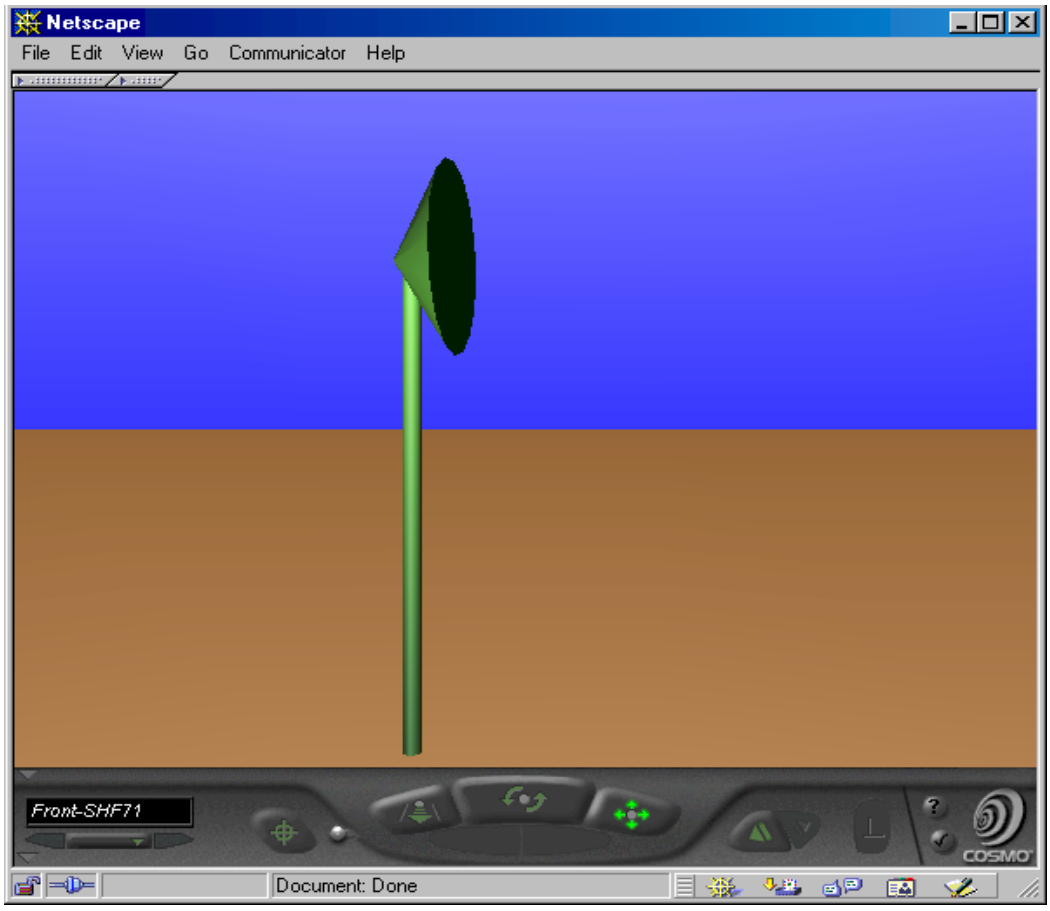


Figure 2.6: SHF MSE Antenna in VRML World (Cylinder and Cone Nodes).

This program reads a file and converts the "post" height data in a VRML *IndexedFaceSet* node, which then can be viewed in a web browser. The *IndexedFaceSet* node provides a mechanism for creating complicated, realistic shapes in VRML. Once calculated, the object is placed as a value in the geometric field of the *Shape* node and can be manipulated like any primitive shape. *Shape* nodes are used to build primitive geometries, which can then be integrated into more sophisticated scene graphs.

The *Shape* node also contains an *Appearance* node that presents the author with intricate control over the color of an object. The *Appearance* node is used in conjunction with the *Material* node and *Texture* node to apply colors and texture images to an object. A terrain map or image can be placed over the elevation data in a virtual world to present a realistic setting (Brutzman, 1998).

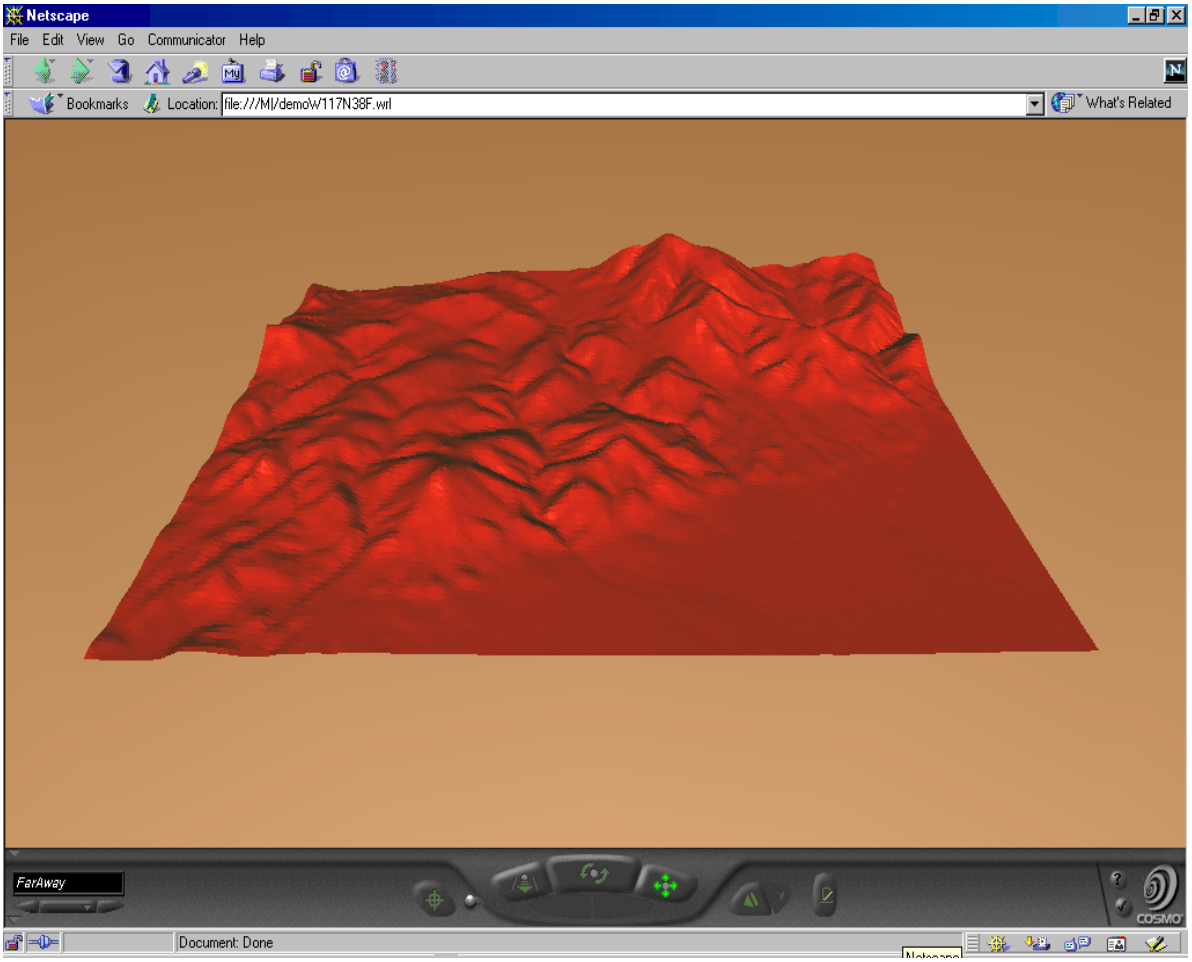


Figure 2.7: 2Km by 2Km VRML IndexedFaceSet of Fort Irwin California generated from NIMA DTED data.

b. Grouping Nodes

Grouping nodes combine sets of nodes for the purpose of creating entire objects, which can in turn be manipulated as another scene graph. These nodes are grouped in a way that makes sense to the viewer of the scene, usually describing a spatial or logical relationship. Grouping nodes are the *Group* node, *Transform* node, *Billboard* node, *Collision* node and the *LOD* (level of detail) node. The most basic of these nodes is the *Group Node*. The *Group* node simply identifies subordinate nodes, which are to be

collected together in a fashion similar to gathering all the parts of the geometry of an object. These subordinate nodes are referred to as the children of the *Group* node. The *Transform* node is a bit more flexible, which makes it a critical grouping node. Like the *Group* node, the *Transform* node not only brings geometric parts together but also is capable of moving the grouped nodes within a local coordinate system. The *Transform* node can specify translational and rotational changes of position to children of the nodes. This represents a fundamental spatial capability of a scene graph. The *Transform* node can be combined with *Interpolator* nodes to create animation in the virtual world (Brutzman, 1998). *Interpolator* nodes define a piecewise-linear function f(t) on the interval (0, 1). A variety of *Interpolator* outputs are available for producing positions, orientations, colors, etc. The piecewise-linear function is defined by n values of t, called key, and the n corresponding values of f(t), called keyValue. The cycle of the *TimeSensor* driving normalized *Interpolator* key input determines the duration of the interpolator period.

c. Viewing Nodes

The Viewpoint node defines a specific location in the local coordinate system from which the user can view the scene. It also allows a scene graph to include predefined camera angles. A set of carefully designed viewpoints can make navigation of a virtual world easier for the viewer. These predefined views of the world are easy to access via the interface bar of the VRML browser offering practical navigation. By having predefined viewpoints, exploration is faster in large virtual worlds, thus enabling

the user to go to the correct point in the scene quickly. Viewpoint information can also be

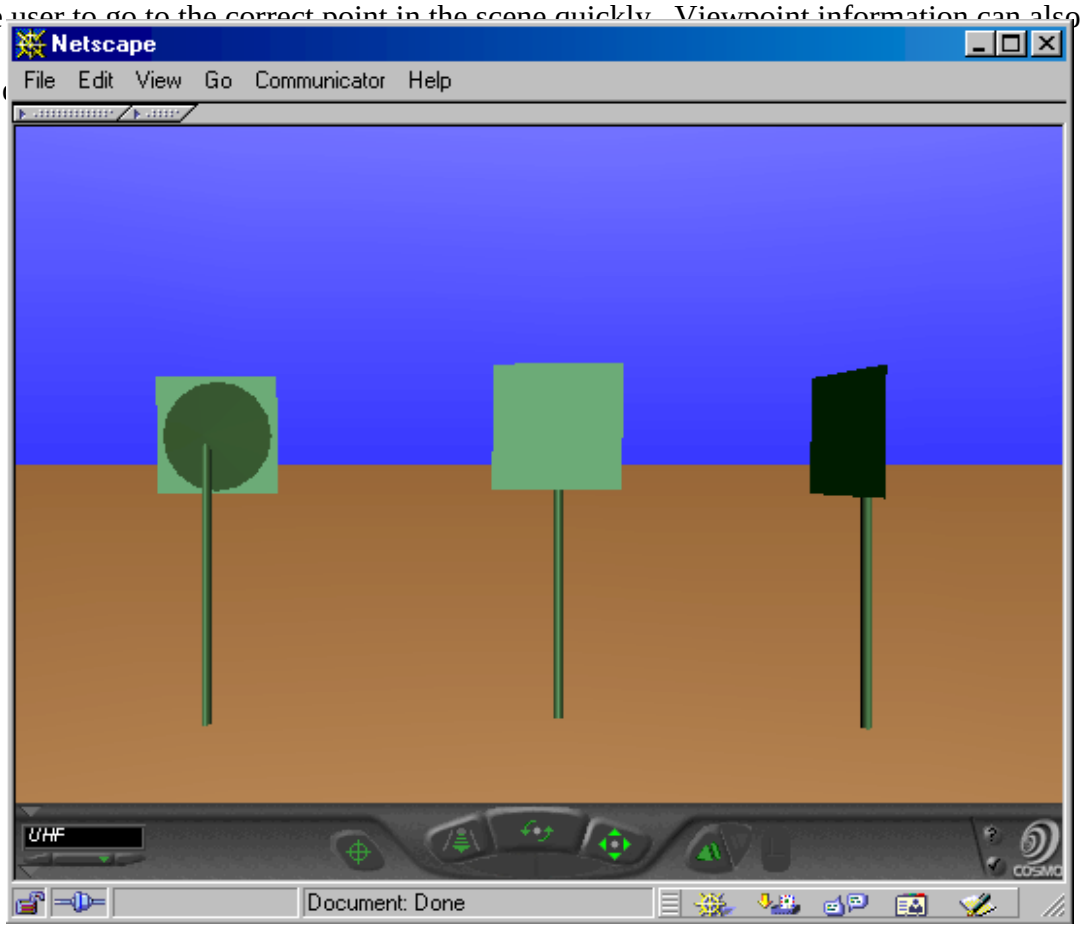


Figure 2.8: MSE UHF antenna using Transform Grouping of Cylinder, Cone and IndexedFaceSet.

In the Cosmo Software VRML Plug-in displays (shown above), viewpoint information is at the bottom right of the screen. The related *NavigationInfo* node permits a virtual world to control how a viewer moves about the scene. A viewer may be forced or guided to walk (rather than fly) through a scene (Brutzman, 1998).

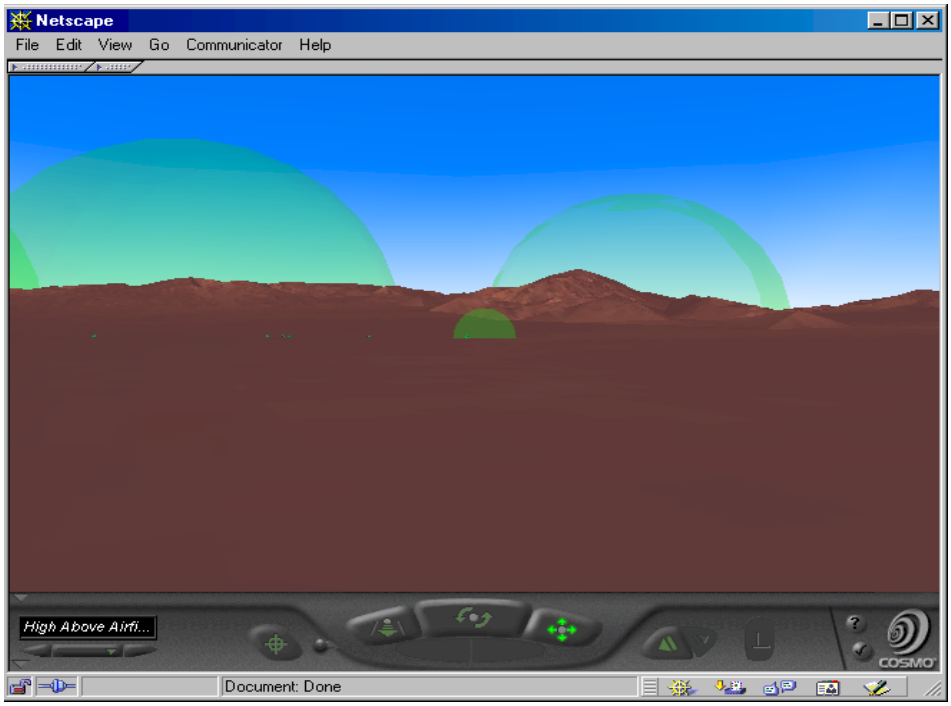


Figure 2.9: View Point high above airfield with communication coverage area shown as green half domes.

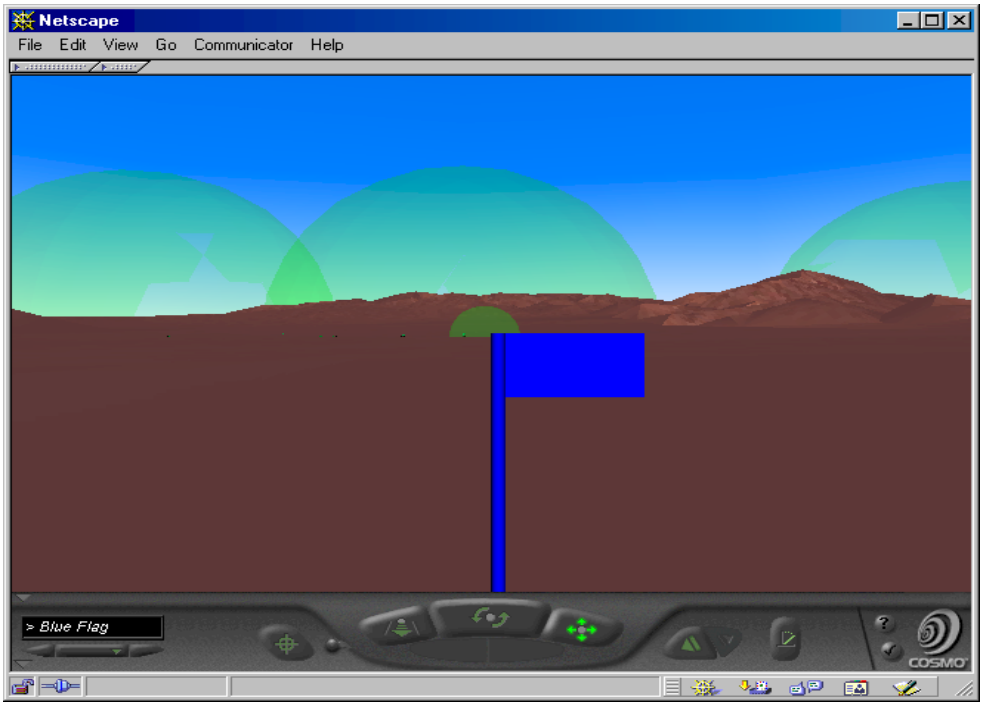


Figure 2.10: Viewpoint next to Blue Flag with communication coverage.

d. Interpolators and Route Nodes

After the geometry and rendering for virtual objects are developed and positioned in a scene graph, it is often necessary to animate these objects. Animation in VRML is accomplished using interpolators and routes. Interpolator nodes are designed for linear key-framed animation. An interpolator node defines a piecewise-linear function f(t) on the normalized time interval (0, 1), and continuously returns a series of values of interest. The 3D browser calculates interim states between key values so that the animation transitions smoothly from the start state to the end state. The most common interpolators are the position and orientation interpolators. These interpolators are used to create translational and rotational animation of objects in the scene graph. Interpolators are available for most base types, and their functionality can be furthered extended by using *Script* nodes.

After these calculations are completed, the values must be passed as events into the *Transform* node of the object for the movement to take place. This value passing is done via *ROUTEs*, which dispatch events from one node to the next node. For example a *ROUTE*, a connection between VRML nodes, can take the calculated changes from an interpolator and redirect the data into the *Transform* node. The modified field in the *Transform* node implements the behavior changes in the scene graph. (Ames, 1997) This thesis work uses event passing to adjust the radius of the communications domes and position of the antennas on the virtual battlefield.

The prerequisite node for driving an animation process is the *TimeSensor* node. The *TimeSensor* node provides the clock that the interpolators use to

systematically output positional data. In this signal visualization, *TimeSensor* is used to adjust the frequency of the number of beam cones sent from one directional antenna to another directional antenna.

e. Sensor Nodes

There are six types of sensor nodes in the VRML specification. These sensors are the *TimeSensor*, *VisibilitySensor, TouchSensor, PlaneSensor*, *SphereSensor*, and *CylinderSensor*. Sensors provide the primary means for a viewer to interact with a virtual world (Brutzman, 1998). They are trigger-based on either time or user intervention. The visualization presented in this thesis uses both time and user input to control antennas on the virtual battlefield. A *TouchSensor,* which activates whenever a mouse cursor or pointing device is placed over or clicks on an object, detects when to display DIS information on the antennas. Sensor functionality can be further extended using *Script* Nodes.

f. Script Node

The VRML *Script* node is used to integrate imperative programming languages such Java and JavaScript (formerly known as EcmaScript) into the scene graph. The *Script* node is used to connect programmed behaviors into a scene. *Script* nodes typically signify a change or user action, receive events from other nodes, and contain a program module that performs some computation, thereby effecting change somewhere else in the scene by sending events. (VRML SPEC, 1997) The *Script* node adds the flexibility needed to develop more sophisticated scene-graph behaviors not inherent in the VRML specification. This thesis visualization uses the *Script* node to calculate the size of the transmission cones and radio coverage domes. The *Script* node is

frequently used to perform network access or physics calculations such as those needed by VRML interpolators and sensors.

g. PROTO and EXTERNPROTO Definitions

The *PROTO* and *EXTERNPROTO* definitions are used to effectively create new VRML nodes as combinations of other predefined VRML nodes. This technique is useful when developing large, specialized scene graphs for objects or models that are needed in a virtual world. *PROTOs* can be used to construct complex objects and behaviors that are referenced either multiple times or in the same VRML file. A *PROTO* defining an antenna needs only be built once, and then it creates instances whenever the user needs that type of antenna. *PROTOs* are a key mechanism for efficiently creating large and intricate virtual worlds. In order to achieve efficient code re-use, the *EXTERNPROTO* construct is provided to group *PROTOs* into libraries in files, which are external to the main scene. This permits storing geometry of complex objects or models on a local hard drive or on a network-accessible storage device for shared access.

2. GeoVRML

The VRML specification is a powerful tool for representing 3D worlds. However, the specification does not directly represent or utilize geographic concepts such as latitude/longitude coordinates or the corresponding navigational movement associated with these coordinate systems. GeoVRML 1.0 (www.geovrml.org) defines geo-referenced scene-graph objects and data (such as maps and 3-D terrain models), to be viewed over the web by a user with a standard VRML plug-in for their web browser. This work was initially developed at the Stanford Research Institute (SRI) and is now

available to the public. SRI and the GeoVRML working group developed a large set of VRML nodes and software tools to simplify implementing geographic constructs in VRML97 specification. The key components of GeoVRML are the *PROTO* nodes designed for referencing and interpolation of virtual worlds through geographic mechanisms. The GeoVRML *PROTO*s use underlying Java code and Script nodes to perform the physically based calculations and perform geographic modeling. The GeoVRML suite is a “Recommended Practice” of the Web 3D Consortium. (Iverson, 1999)

GeoVRML nodes perform functions similar to the corresponding nodes defined in the VRML97 specification. The left side of Figure 2.11 shows a GeoVRML code fragment used to rebuild the geo-referenced virtual scene displayed in the right side of Figure 2.11. While this example code is missing a large number of VRML declarations needed to provide geo-referencing, the major nodes required to render the scene are present. The virtual Earth appears to be similar to the "Hello World" Earth of Figure 2.5 but it is more sophisticated. The two virtual worlds provide similar visual representation of the Earth, but the original "Hello World" Earth is a sphere wrapped with a texture map of the Earth. The Earth built in Figure 2.11 is composed of elevation grids geo-referenced by latitude and longitude. This provides the planner with a way to correctly represent the coordinates that they use to plan their areas of radio coverage in the virtual world by using geographical coordinate systems corresponding directly to the Military Grid Referencing System (MGRS) (TB 11-5895-1544-10-1).

GeoVRML provides a solution to the lack of geo-referencing in VRML. The ability to convert the raw digital terrain elevation data (DTED) post height information provided by NIMA into correctly geo-referenced maps for the virtual world allows the planner the flexibility to plan a mission for any type of terrain. The rapid availability and ease of access to DTED data ensures that it is possible to achieve minimal delay between the initial plan and its visualization on a geo-referenced virtual map.

a. GeoOrigin

Coordinate reference systems currently supported by the GeoVRML suite assume that the virtual world begins at the center of the earth. In order to gain optimal precision of the model, GeoVRML provides the *GeoOrigin* node to specify the local coordinate system. Only one *GeoOrigin* node is used within a scene. It directs the browser where to look inside the VRML world and to interpret the geologic data (Reddy, 2000).

```
GeoViewpoint {
   geoOrigin IS geoOrigin
   geoSystem ["UTM" ,"Z11"]
   position "3905500 578200 10000000"
   orientation 1 0 0  -1.57
   description "Welcome to Monterey"   }
#Build Earth w/ GeoElev. Grid & Lat/long
  Shape {
     appearance Appearance {
      material Material {
       diffuseColor 0.8 1.0 0.3 }
       texture DEF TEX ImageTexture
                          { url "earth.gif" }}
     geometry GeoElevationGrid {
            geoOrigin IS geoOrigin
            geoSystem [ "GDC" ]
            geoGridOrigin "-90 -180 0"
            xDimension 21
            zDimension 21
            xSpacing "18"
            zSpacing "9"
            height [0 0 0 0… (441 total) ]}}
GeoLocation {
     geoSystem [ "GDC" ]
     geoCoords "36.601388 -121.88166 200000"
#----------Lat/Long Monterey CA
     children [
       Transform {
         rotation 1 0 0 3.1415926
         children [
           Shape {
            appearance Appearance {
              material Material {
                     diffuseColor 1 0 0 }}
           geometry Cone {
                     bottomRadius 100000
                           height 500000 }
} ] } ]  }
```

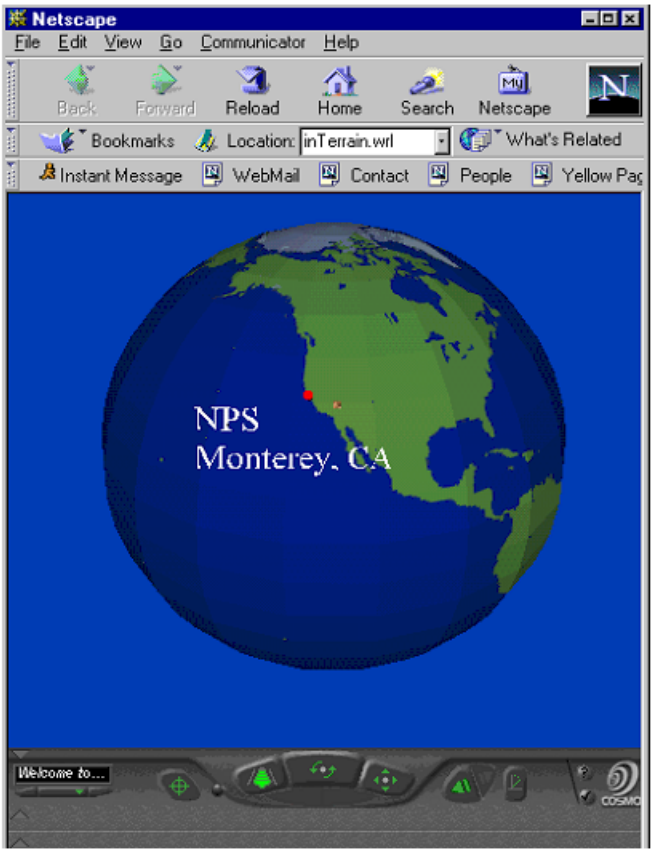


Figure 2.11: “Hello World” source code and rendered scene redone in GeoVRML.

b. GeoLocation

The ability to place objects in specific locations of a scene graph is of fundamental importance in the military, where maps are used in the planning of every mission. The *GeoLocation* node presents the capability to place objects in a virtual world using a geological reference frame. This node performs in a manner similar to the *Transform* node of VRML97 specification (Reddy, 2000).

c. GeoPositionInterpolator

The original *Transform* node required the *PositionInterpolator* node to create smooth movement for animation. The *GeoLocation* node also has an associated interpolator. The *GeoPositionInterpolator* node performs the function of calculating key

values and intermediate positions in geographical coordinates (Reddy, 2000). The ability to move and update the location of an antenna in the scene is required as the virtual battlefield changes. The ability to record and to position changes in the simulation helps the planner who is working to plan an active mission.

d. GeoViewpoint

The *GeoViewpoint* node behaves like a standard *Viewpoint* node. The *GeoViewpoint* node relocates the viewer's orientation and position to an absolute posture in the geo-referenced coordinate frame. The *GeoViewpoint* node supplies a practical means for maneuvering about complex GeoVRML worlds.

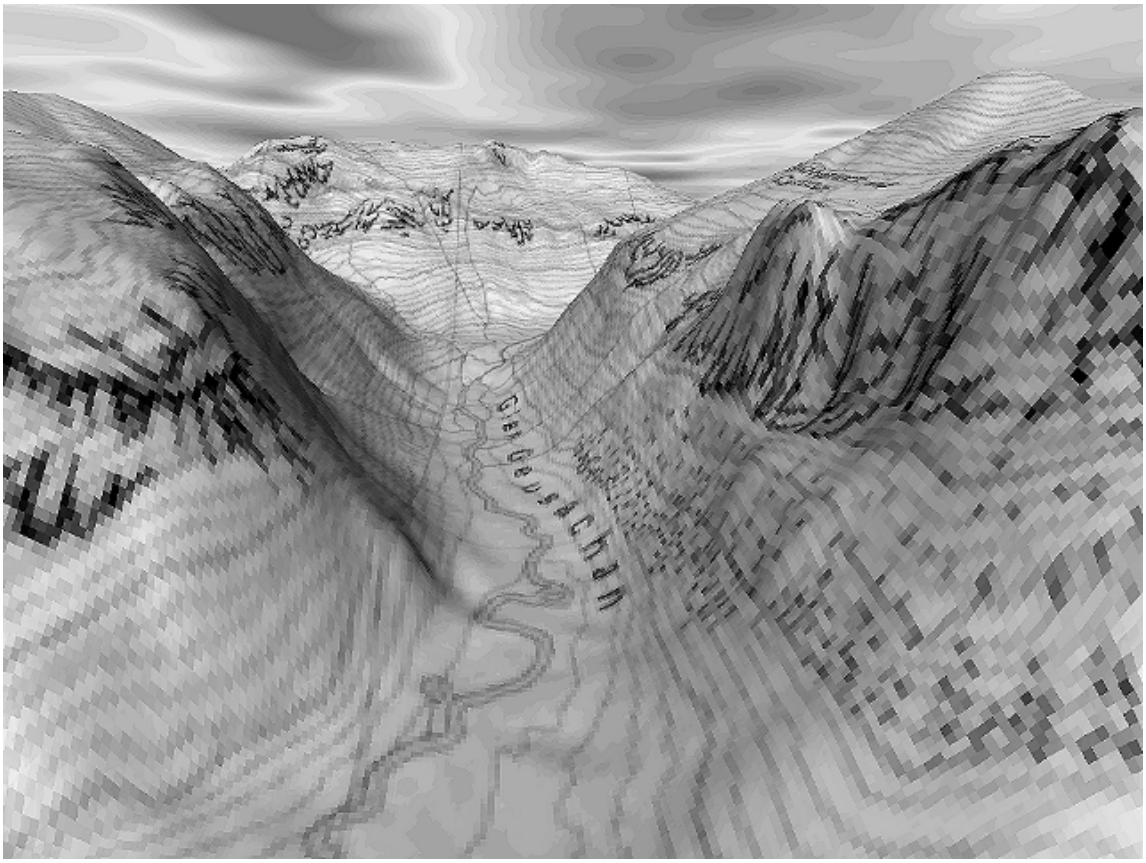


Figure 2.12: Generated VRML terrain of Valley in Cairngorms (vertically exaggerated) with an overlay of 1:25000 Map which was in GIF format Form Ref (McCullagh 1997).

I. DIS-Java-VRML

DIS-Java-VRML is a tool kit that provides DIS connectivity with the integration of the Java™ programming language, and VRML 3D modeling capabilities. It provides the developer with a set of libraries and examples in which they can develop networded 3D virtual worlds. "DIS-Java-VRML is an open source software toolkit and library of applications that enables VRML 3D scenes to be DIS compliant through the use of Java networking code" (Brutzman 2000 email). The DIS-Java-VRML working group's goal was not to invent a new protocol design but to develop a Java class library and architecture for exchanging DIS packets over the Internet. Through the use of the script nodes in VRML, the Java classes use the event in and event out fields in VRML to update the scene. A developer using DIS-compliant code ensures that his or her simulation will operate with other DoD Simulations. The connecting of 3D virtual worlds with networking provides an optimum environment for a 3D collaborative planning tool. This tight integration of the DIS networking code with VRML presents an opportunity for a web-based virtual battlefield. A shared virtual battlefield with commanders and planners at different locations benefits from this type of capability.

> "DIS, Java and VRML can provide all of the pertinent capabilities needed to implement large-scale virtual environments (LSVEs). DIS is essentially a behavior protocol tuned for physics-based (i.e. "real world") many-to-many interactions. Java is the programming language used to implement the DIS protocol, perform math calculations, communicate with the network and communicate with the VRML scene. VRML 3D graphics are used to model and render both local and remote entities in shared virtual worlds." (Brutzman 2000 website)

DIS-Java-VRML development software and models are available at (http://www.web3d.org/WorkingGroups/vrtp/dis-java-vrml).

J. EXTENSIBLE 3D (X3D)

The next-generation VRML specification is referred to as Extensible 3D (X3D). Extensible 3D contains the XML encoding of VRML97 specification. By using XML, the new X3D standard includes constructs a document type definition (DTD) tagset that allows users to develop well-formed and validated scene graphs. XML also provides X3D with extensibility, meaning the ability to define and integrate new nodes at runtime. Extensible 3D has fundamental nodes and structures similar to VRML97 standard and is fully backward compatible.

X3D allows users to create geometries that contain metadata. Metadata is data that describes the contained geometries to other applications. The rapid encoding of geometries and the ability to define new geometries, ensures that the most accurate representations are rendered correctly. The utilization of the metadata is powerful because it provides the communications planners with definable geometries and the ability to transform, organize, and render data in ways that suits their needs.

Using an X3D software development kit and the X3D-Edit authoring tool, developers can produce validated scene graphs with error-free editing. This tool utilizes IBM's Xeena XML editor, which has been configured to facilitate straightforward development of scene graphs that conform to the X3D DTD. The X3D-Edit tool converts X3D documents to VRML97 via an Extensible Stylesheet Language (XSL) stylesheet and automatically launches a browser for convenient debugging. Figure 2.13 shows a screen capture of X3D-Edit (Extensible 3D Task Group, 2000).

Extensible 3D provides the critical link between the XML documents and visualization presented in this thesis. Although VRML97 is the basis for many of the models developed, X3D provides the structure and flexibility to transform XML documents to valid scene graphs. Figure 2.15 and Figure 2.16 show examples of X3D and VRML code, which render identical virtual worlds. Using a X3D/VRML approach provides interoperability between web-based simulations.

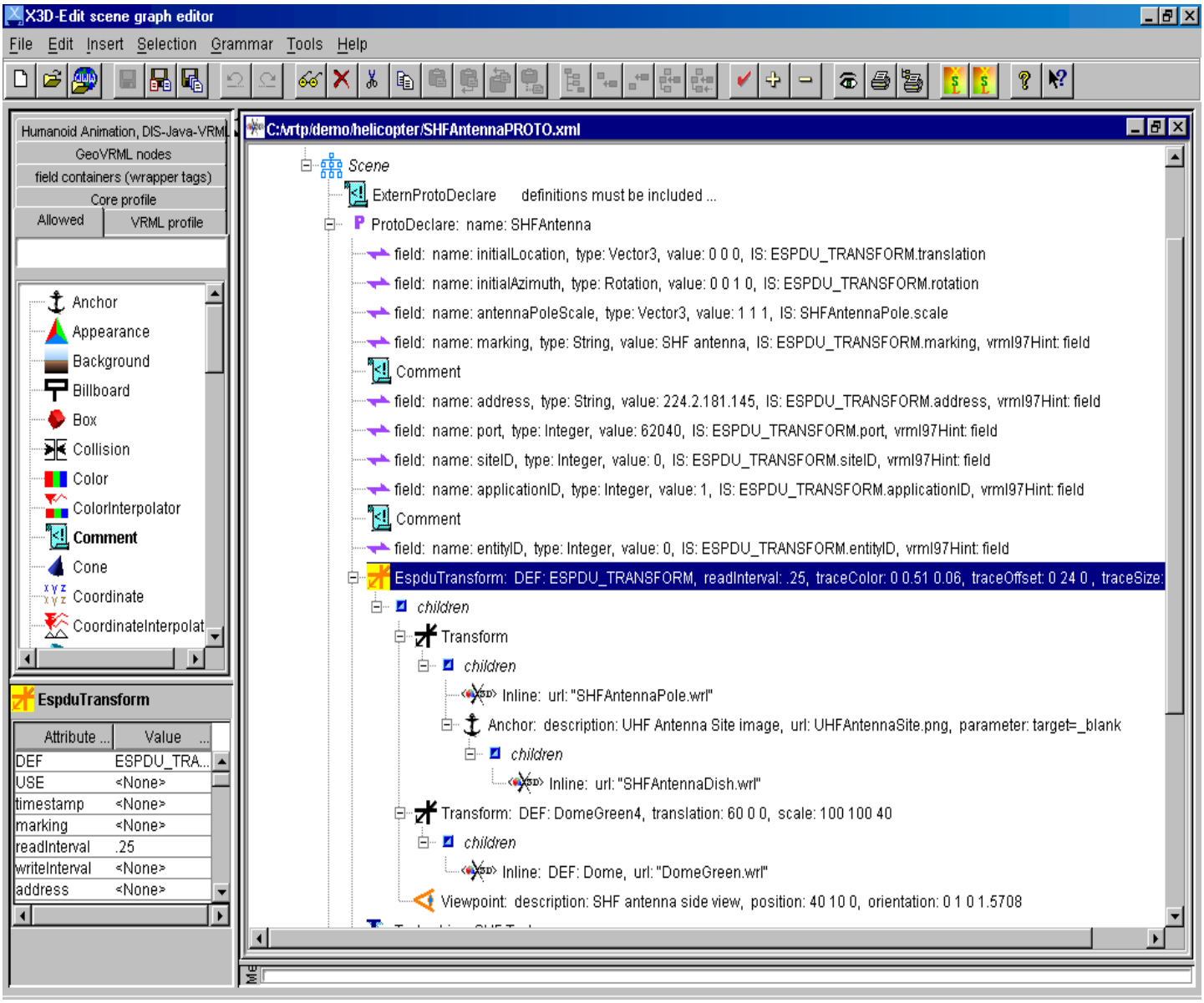


Figure 2.13: Screen capture of X3D-Edit Tool with allowed-node menu.

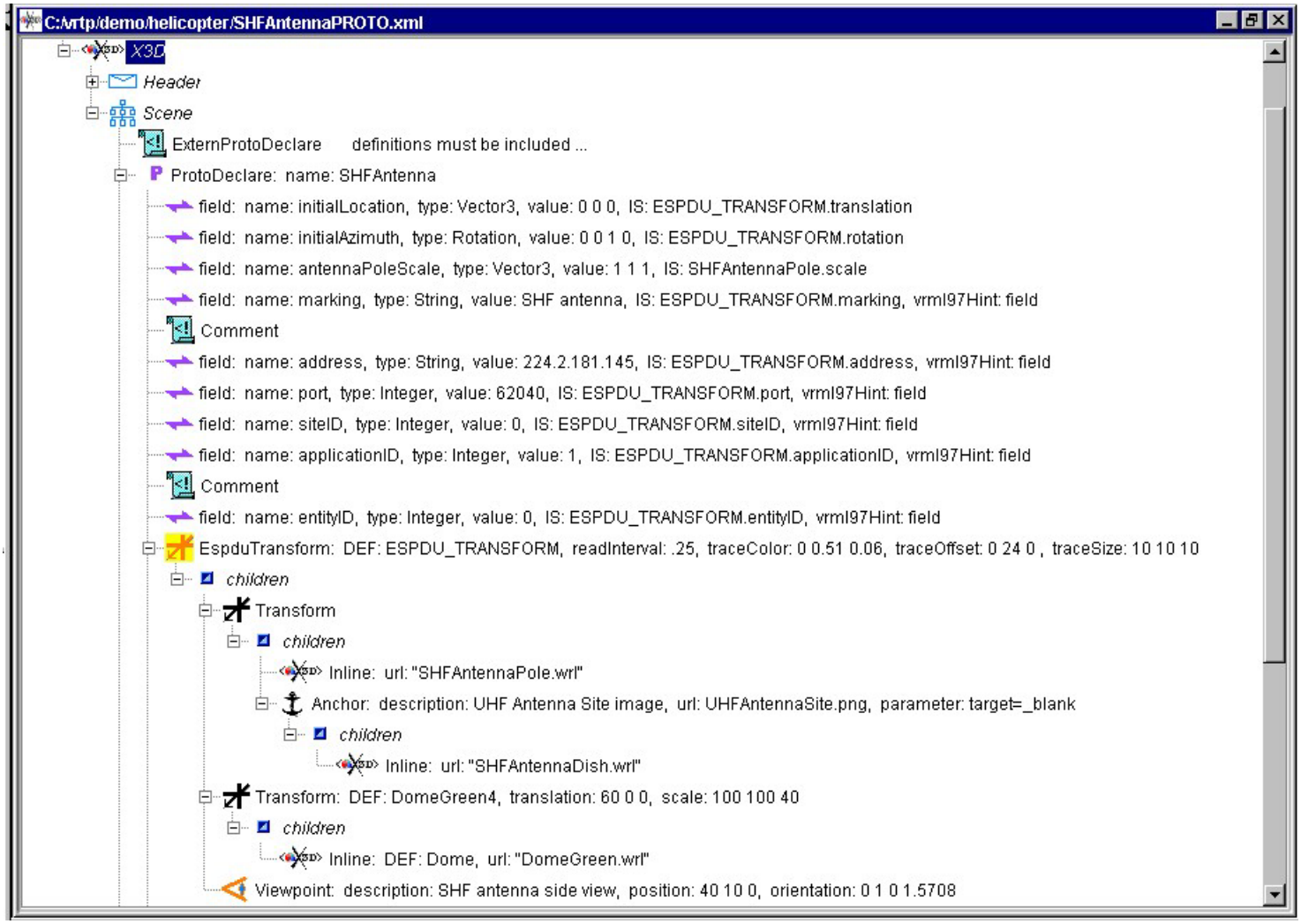


Figure 2.14: Screen capture of X3D-Edit Tool showing SHF Antenna Prototype.

```
#VRML V2.0 utf8
Group {
 children [

   Viewpoint {
        description "initial view"
        position    6 -1 0
        orientation 0  1 0 1.57 }

   Shape {
        geometry    Sphere  { radius 1 }
        appearance  Appearance{
            texture ImageTexture {
                    url "earth-topo.png"}}
         }

   Transform {
     translation 0 -2 1.25
       rotation    0  1 0   1.57
       children [

           Shape {
             geometry  Text {
              string [" Hello" "world!" ]}
             appearance Appearance {
               material Material {
                  diffuseColor 0.1 0.5 1 }}

                }
               ]
             }
           ]
         }
```

Figure 2.15. X3D Source Code for "Hello World".

```
<X3D>
  <Scene>
     <Group >
       <Viewpoint
            description='initial view'
            orientation='0.0 1.0 0.0 1.57'
           position='6.0 -1.0 0.0'
        />
       <Shape>
           <Sphere radius='1.0'/>
               <Appearance>
                  <ImageTexture
                    url='"earth-topo.png" "
                  />
               </Appearance>
       </Shape>
       <Transform
            rotation='0.0 1.0 0.0 1.57'
            translation='0.0 -2.0 1.25'>

         <Shape>
           <Text string='"Hello" "world!"'/>
           <Appearance>
             <Material
                diffuseColor='0.1 0.5 1.0'
             />
           </Appearance>
         </Shape>

       </Transform>

    </Group>
  </Scene>
</X3D>
```

Figure 2.16. VRML Source Code for "Hello World".

K. TOOL REQUIREMENTS

While researching this thesis, the author preformed a literature survey to locate a software toolkit that might help develop a system based on a common open standard for data interchange. This system is intended to allow the 3D visualization of radio profile information, overlaid on a realistic terrain model, to be shared among many participants on open or classified networks. Open standards and high-quality content-development authoring are the primary tool requirements.

L. COMMUNICATIONS VISUALIZATION TOOLS

Three major software tools are being used currently to model communications networks within the US Army Signal Corps. These tools are OPNET Modeler™ by MIL 3 Inc (MIL 3, 1987), COMNET™ III from CACI Products Company (COMNET, 2000), and the MSE-NPT (MSE-NPT, 1997) software developed under contract for the U.S Army Communications Electronics Command by General Telephone Equipment (GTE) Corp. These tools each have unique capabilities. The focus of this thesis is on each software suite's ability to perform radio profiling and produce three-dimensional (3D) visualizations of radio wave propagation.

1. OPNET Modeler™

OPNET Modeler™ by MIL 3 Inc. is a computer-aided system for the design, simulation, and analysis of communications networks, computer systems, applications and distributed systems. MIL 3 Inc. introduced OPNET in 1987. Over 500 civilian and

Department of Defense (DoD) organizations are currently using OPNET™ (MIL 3, 1987).

OPNET™ provides the user with the ability to model from the Wide-Area Network (WAN) down through the process and state level. Various degrees of modeling resolution provide the user with the added flexibility to model current or design future communications and computer networks, systems and applications. OPNET™ communications modeling software was selected by the J-6 NETWARS program to develop future communications packages for the Army. OPNET™ is a commercial off-the-shelf (COTS) program which can be installed on a number of platforms: Sun SPARC Solaris 2X; Sun SPARC Sun OS; HP UNIX and Intel-based implementations of Windows™ NT and Windows™ 2000.

OPNET™ can model multiple client/server applications (email, database, file transfer, etc.), peer-to-peer delays, server backlogs, and transaction response times and throughputs. OPNET™ can also models satellite systems and mobile communications nodes, but it does not focus on radio wave propagation. Some military systems libraries are available for the product, and these can be obtained from various DoD agencies. This system costs $18,000 per year and is used at the strategic and engineering levels. OPNET™ network modeling requires a programming background and a solid understanding of statistical methods as well as a complete understanding of analog and digital communications. OPNET™ uses a proprietary method for the storage of information via its relational database management systems (RDMS).

2. COMNET/STK

COMNET™ III from CACI Products Company is a network planning tool designed with an object-oriented environment to model Local-Area Networks (LANs), Wide-Area Networks (WANs) and Metropolitan-Area Networks (MANs). Network models are created graphically and require no actual programming by the user (COMNET, 2000).

Network traffic files can be imported from various systems. Animation provides the user with the ability to monitor the network flow. COMNET III ™/STK uses familiar objects like computer nodes, routers, and links which can be edited by the user to design a specific network topology. The basic steps used to build a model using COMNET III ™ are to define a topology and establish traffic and computer loads. Reports selected by the user are automatically generated during the simulation run.

COMNET III™ software is available for Windows™ 98, Windows™ NT, Windows™ 2000, all major UNIX computers.

Wide-Area Networks (WANs) and Local-Area Networks (LANs) are easily modeled through the use of subnets and WAN cloud icons. These provide the user with various views of the network hierarchy. WAN clouds can be used to model frame relay, cell relay (ATM) and packet switching (X.25). Packet switching (X.25) is the type of switching most common on today's battlefield in Mobile Subscriber Equipment (MSE) communications shelters.

COMNET III™ provides radio communication modeling through the use of an add-on package called the Satellite Tool Kit (STK) ™ from Analytical Graphics, Inc. STK™ provides radio and radar modules and lets users select from among several single-beam antenna types, then define and orient the type selected. In addition, the radio module lets users specify wave polarization parameters.

COMNET III™ supports single-beam antennas and multi-beam antennas and includes properties for polarization and antenna orientation. Users can visualize communications system in both 2D and 3D. Figure 2-17 is a 2D representation, which presents the coverage in a 2D projection view of terrain with satellite wave propagation, and Figure 2-18 is a 3D representation, which shows the area coverage for a satellite system in space.

COMNET III™ and STK provide for networked visualization through the use of the DIS Protocol. STK/DIS module is a DIS software application designed to create complex virtual scenarios for communications systems simulation. The STK/DIS module connects with the DIS to read protocol data units (PDU) and then populate a shared scenario. The PDU Entity State data (position, attitude and dead-reckoning algorithmic parameters) is used to define the state of the STK object. The resulting scenario is a complex virtual-reality planning tool that is accurate and up to date. STK/DIS exercises are used by the military for training, test and evaluation, and concept analysis.

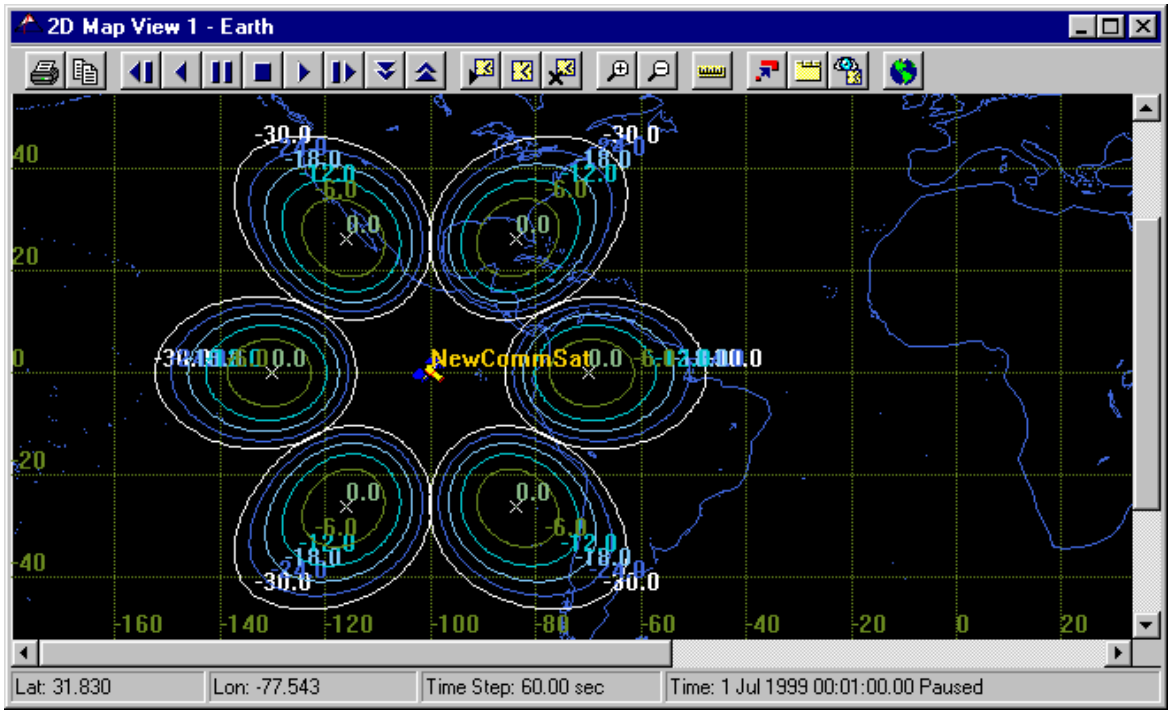


Figure 2-17: Satellite Tool Kit (STK) screen snapshot of 2D radio profiling.

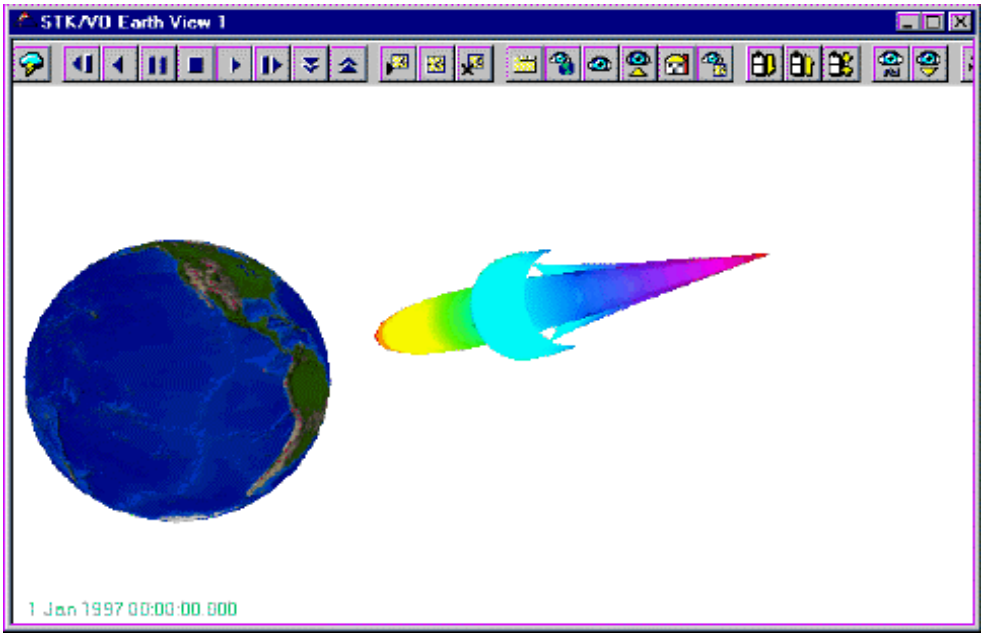


Figure 2-18: Satellite Tool Kit (STK) screen snapshot of 3D satellite profiling.

This software system costs $48,500 and is used at the local installation engineering departments. Use of COMNET III™ requires a programming background and a solid understanding of statistical methods, as well signaling and digital communications. The Satellite Tool Kit (STK) package is recommended by the Communications and Electronics Command (CECOM) for modeling space-based communications platforms. COMNET III™ uses its own proprietary method for the storage of information with its RDMS.

3. MSE-NPT

The Mobile Subscriber Equipment (MSE) Network Planning Tool (NPT) is a set of radio-profiling software and hardware that was developed under contract for the CECOM (MSE-NPT, 1997). The MSE-NPT is a Modified Table of Organizational Equipment (MTOE) item for all communications units that have MSE transmission equipment. This radio-profiling tool is issued to all units that have radio transmission equipment, and the field version of the NPT is customized for MSE equipment. Major components of MSE include node centers (NCs), radio access units (RAUs), large extension nodes (LENs), small extension nodes (SENs), mobile subscriber radiotelephone terminals (MSRTs), and system control centers (SCCs).

NPT is a network-planning tool designed with an object-oriented software architecture to engineer radio systems in the MSE shelters. One such radio is the AN/GRC-224 SHF radio system operating in the 14.50 to 15.35 Gigahertz (GHz) range. They are found in the AN/TRC-190 LOS radio terminal shelters used for short-range communications from 2 to 5 kilometers. MSE-NPT also engineers systems utilization on

the AN/GRC-226 UHF radio system operating 225.0 to 400.0 Megahertz (MHz) (BAND I) and 1350.0 to 1850.0 MHz (BAND III) ranges. These radios are found in the AN/TRC-190 LOS radio terminal shelters used for long-range inter-nodal communications from 20 to 50 kilometers. MSE-NPT can also engineer systems based on the AN/VRC-97 VHF MSRT radios. The AN/VRC-97 radio has a tuning increment of 25 KHz and can shift automatically to find an unused channel for communications. The AN/VRC-97 works much like a spread-spectrum cell phone.

A user-designed NPT network is created using a 2D graphical user interface (GUI) and requires no actual programming by the user. A good knowledge of reliable link-performance fundamentals is required. The operator must have an understanding of the specific capabilities radio for each interface type. The MSE-NPT uses familiar modeling objects based on standard military symbology for communications shelters, links between shelters and tactical units. The basic steps to build a model using MSE-NPT including: defining the communication nodes with their geographic locations, profiling the terrain from transmitter to receiver, and checking the profile simulation results. The MSE-NPT system does not provide a simulated load on the system. It does provide a detailed analysis and description of propagation loss, fresnel zone clearance, multi-path fading, link margin, path reliability and climate factors.

The NPT does not provide the ability to network multiple planning terminals, which can only be configured to run in a stand-alone fashion. Nevertheless a user is able to load multiple network configuration files into the same terminal to view the networks by overlay. The NPT only uses military-standard NIMA DTED products for its

geographic baseline. The primary output product provides the user with a color 2D view of the network overlaid on these DTED Maps.

The MSE-NPT system does not have a unit cost since it is provided to each Army unit as part of the MSE equipment package. A prerequisite for proper MSE-NPT operation is that the operator attends the Army's Battlefield Spectrum Management Course per the guidance of the Army's Signal Center (http://www.gordon.army.mil).

M. SUMMARY

This background work has provided an overview of the tools and technologies that are used in communications planning. These tools and technologies when connected to a network and integrated can together provide the signals planner with an enhanced visualization capability.

III. METHODOLOGY

A. INTRODUCTION

This chapter describes the design for a system to develop and generate a 3D radio-communications profile plan. This chapter also explains the how technologies discussed in Chapter II relate to these communications-visualization techniques. Later chapters present more specific details on the DIS-Java-VRML implementation.

B. OPPORTUNITY STATEMENT

Signal-communications planning is currently performed by manually plotting antenna positions on terrain maps, indicating the area of coverage on each map using range-fan and radio-profile information. This task requires an individual with a considerable amount of experience in radio planning. Such individuals must be familiar with the deployed radio equipment, and must also understand weather and terrain effects for the particular radio systems that they are planning for. The manual planning method has been used successfully for a number of years. However, as networks combine the use of data and voice circuits, the planner considers a greater number of variables. This growth has made communications planning a complex and difficult problem to understand. It is possible to simplify this overall planning problem by using methods of information visualization and scientific visualization.

C. PROPOSED IMPLEMENTATION

The general design of the 3D communications visualization can be decomposed into five steps, as listed in Figure 3.1. These five steps define the major steps of the implementation. A flow chart illustrating typical workflow is shown Figure 3.2

1. **Input information**: A user performs an initial design of a signal plan using the MSE-NPT system in accordance with current doctrine. Then the generation of the raw MSE-NPT file and terrain locations of antennas with MGRS positions then convert MGRS positions to VRML positions, or MGRS to Northing and Easting.

2. **Separate out the antenna information**: This step is the process of gathering the antenna specific information, like the frequency, polarization, antenna height, azimuth, elevation, and antenna type.

3. **Process the input**: This step is the creation of the Java antenna objects, which have an associated *SignalPDU, ReceiverPDU*, or *TransmitterPDU*. Several examples are provided.

4. **Provide a networked copy of the data**: This step connects the Java antenna objects and their associated *SignalPDU*, *ReceiverPDU*, or *TransmitterPDU* and then broadcast this associated information across the DIS network.

5. **Display the visualization**: This step links of the associated *SignalPDU, ReceiverPDU*, or *TransmitterPDU* data to a graphic representation and displaying it in the scene. Changes can be tested using the Antenna Control panel, then saved back to a MSE-NPT File.

Figure 3.1: Methodology for signal planning-visualization.

This general design pattern is similar to that of DIS-Java-VRML design and leverages previous work of the DIS-Java-VRML helicopter demonstration framework, for visualization of signal communications plans. A more detailed methodology, incorporating many of the technologies discussed in the previous chapter. Figure 3.2 shows a detailed elaboration of the five-step process, including a technological solution for each requirement.

The foundation of a virtual signal communications plan is the MSE-NPT profile data. This data provides the input information needed to start the visualization, which is the first step in the methodology. The MSE-NPT Terminal generates the profile data.

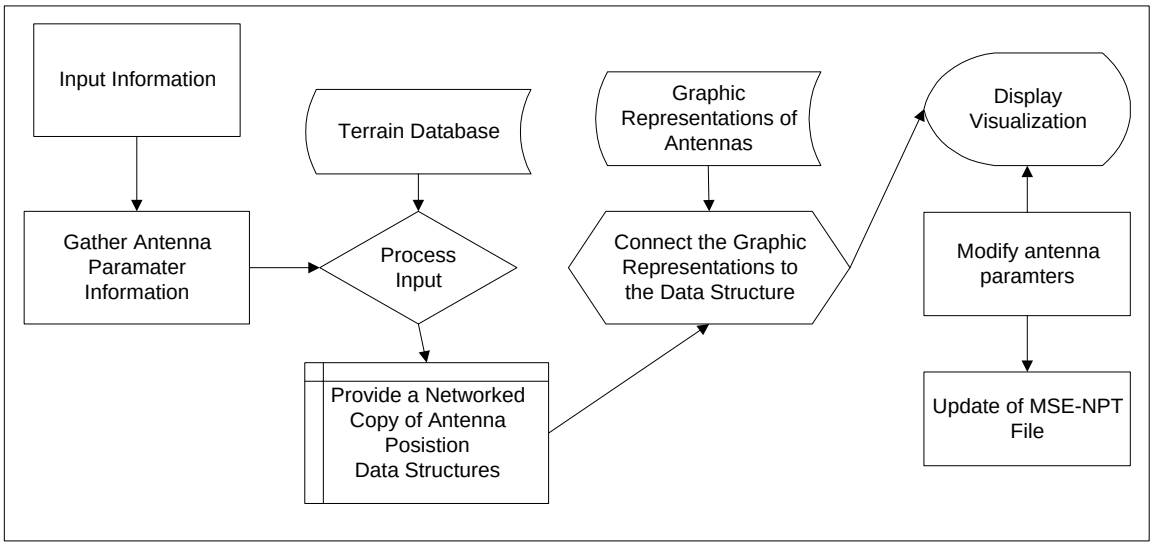


Figure 3.2: Implementation methodology overview for signals-visualization.

After the DIS-Java-VRML framework is started, the MSE-NPT data file is selected and loaded by the user. This file contains a number of parameters about each antenna, such as its position in MRGS. A Java file reader was developed to parse this information from the MSE-NPT data file, perform steps two and three, and transform it into an antenna data structure. The file reader can also parse the MSE NPT numeric data and transformation into DIS PDUs.

The fourth step is to associate the MSE-NPT data structure and the PDUs being sent to it. This is executed within the DIS-Java-VRML framework by connecting the *SignalPDU*, *TransmitterPDU*, or *ReceiverPDU* to the associated unique antenna ID in

the antenna vector. The PDUs are implemented in Java in accordance with the IEEE DIS specification.

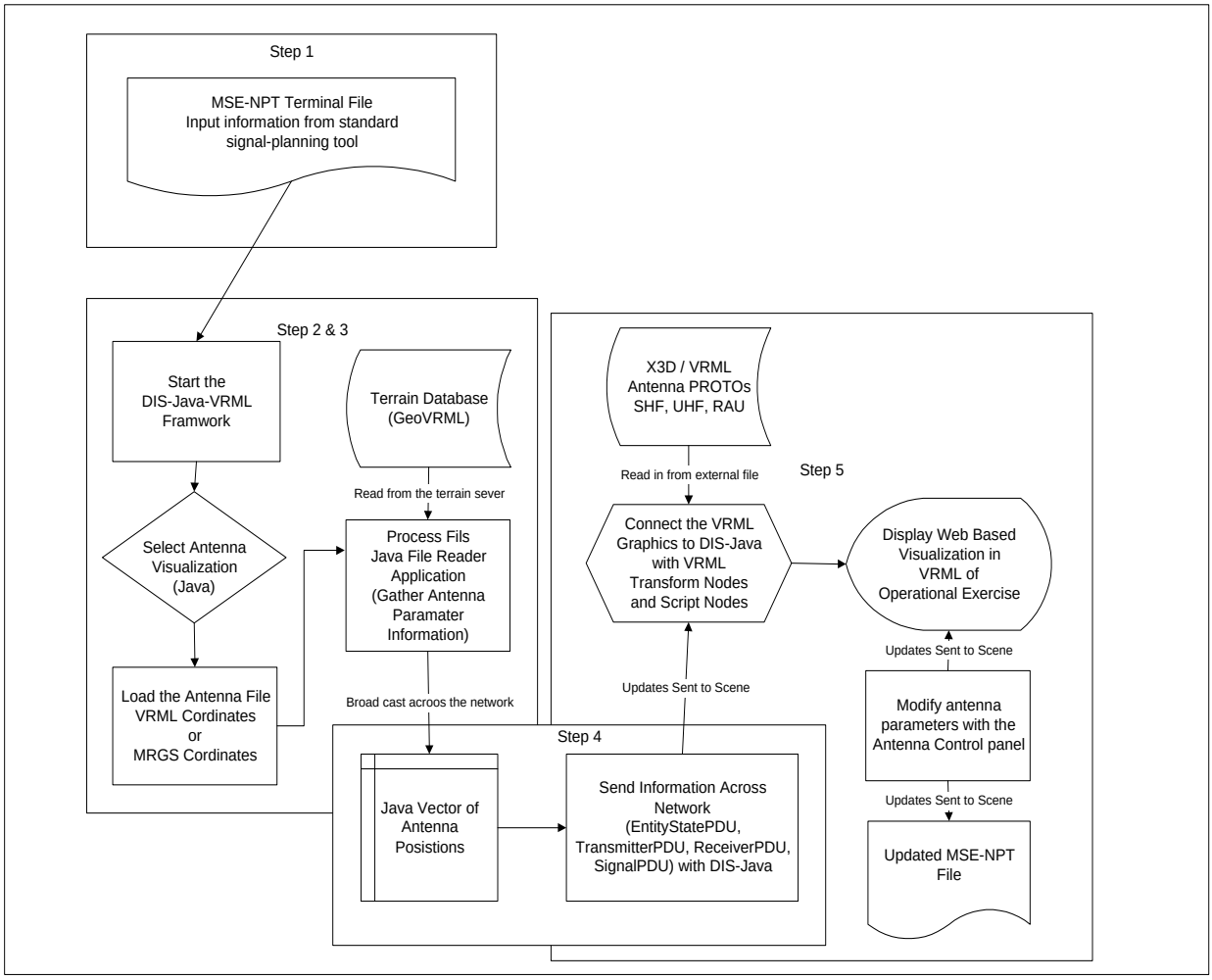


Figure 3-3: Detailed implementation methodology for signal visualization.

The final step is when the PDUs are sent to the DIS-Java-VRML framework via the DIS networking code. By sending the PDUs it will connect the VRML antenna graphical representation to the antenna data structure and then displays the graphical representation of the antenna with the *SignalPDU*, *TransmitterPDU* or *ReceiverPDU* data. Changes to the antennas in the VRML virtual world are done via the Antenna

Control panel. These Java based panels are generated for each antenna instance. The control panels are the users primary means of controlling the positions as well as which PDUs are to be sent to the scene. The graphical representation of the antennas were developed in X3D and then automatically translated into VRML.

D. TECHNOLOGICAL SOLUTIONS

This thesis utilizes a number of the technologies from Chapter II to solve a variety of design challenges. The author chose to use DIS to solve the common network protocol, and to use X3D for the construction of the graphical representations of the antenna, with the Java programming language connecting all the pieces.

1. DIS DESIGN

The DIS network communications protocol has a number of families of PDUs for communications. By using an agreed-upon format for network communications, this visualization should interact with other visualization using this protocol in the same manner. The author implemented Java classes providing DIS Radio Communications family protocols in Java. A detailed discussion is provided in Chapter IV.

2. EXTENSIBLE 3D (X3D) DESIGN

X3D provides the ability to generate the graphic representations of objects and provides interoperability between web-based simulations. The author developed all the graphic representations of antenna as VRML PROTO Nodes. These PROTOs provide the basic rendering geometries of the SHF, UHF, and RAU antenna systems. These

PROTOs can also be translated from their native XML encoding into VRML encoding using XSL style sheet X3dToVrml.xsl that is provided with the X3D-edit authoring tool.

3. JAVA DESIGN

Java is very important to this visualization. Java provides the communication between the *Script* node and the DIS network. This connectivity provides for event passing so that the visualization reflects the current state of the DIS network. There are two aspects to the communication between the *Script* node and the program bound to it. The first concerns the means of using events to actually transfer data from the *Script* node into the Java program. An event-handler Java class file does this. This class in this case is the SignalPDUScript.class for the *SignalPDU*. The second aspect concerns the opposite case how to get the data from the java program back to VRML. This is done with the *Script* node, which is defined in the vrml.node package and provides a mechanism for users to manage the interaction between the VRML nodes and the Java programs (Brutzman 98). By importing this package into a Java program, VRML events can be passed back to the program. The author developed a SignalPDUScriptPROTO that performs this type of event passing, which is shown in Figure 3.3.

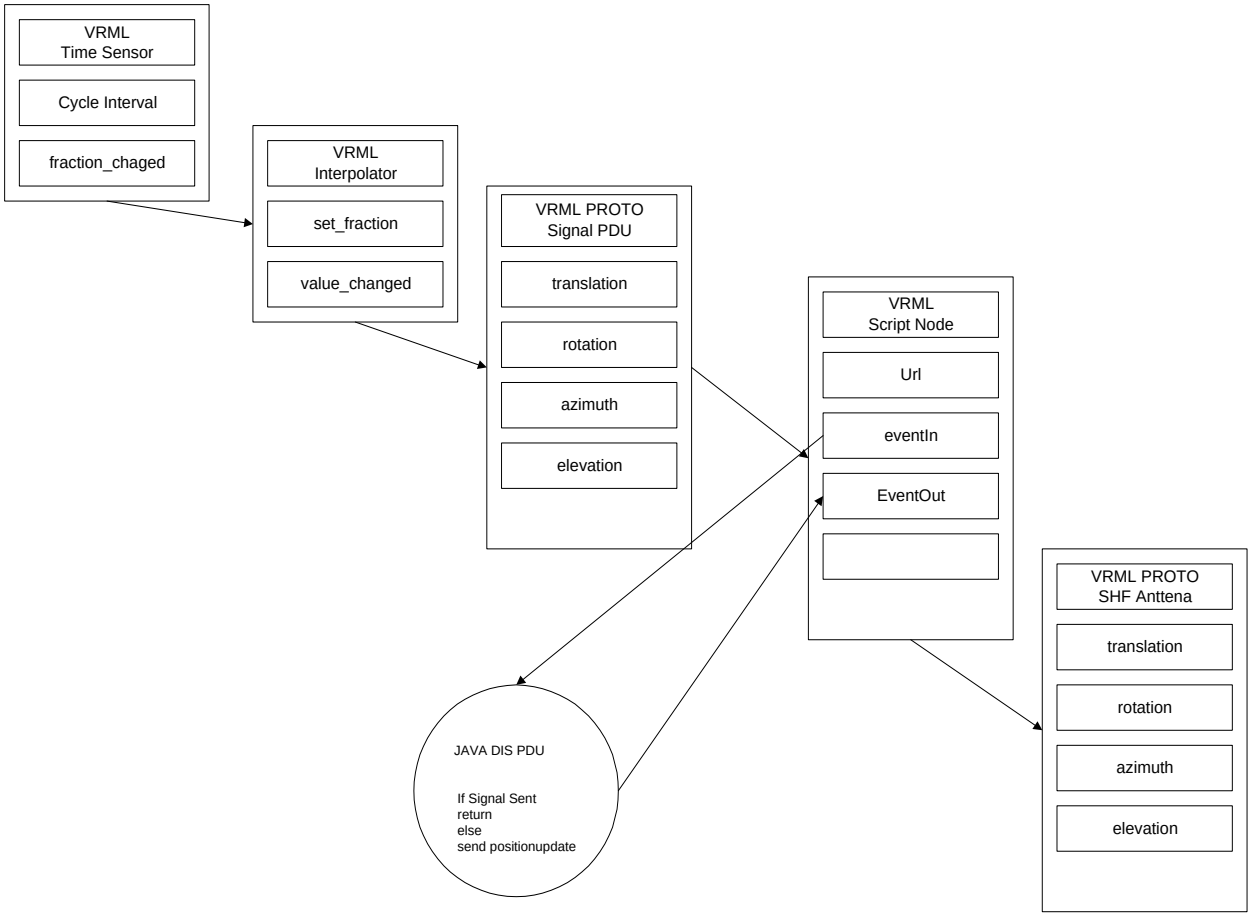


Figure 3-4: Detailed Java connectivity design reads value from the network and sends data to and from the VRML scene.

E. SUMMARY

This chapter provided a design overview. By taking the problem of visualization and breaking it into it smaller parts, each one can be addressed as a part of the whole. This approached allowed the author to approach each with a different technology solution. When these solutions are combined together into the DIS-Java-VRML framework and integrated into a web-based solution. This web-based visualization provides a solution to the difficult problem of the visualization of communications

networks. The detailed implementation of the parts is presented in Chapter VI, Systems Integration and Demonstration.

IV. DIS EMISSIONS AND PDUS

A. INTRODUCTION

This chapter discusses the practical factors involved in implementing the DIS protocol in this visualization. Subjects examined in this chapter include DIS Data Stream PDUs, DIS implementation, and the Radio Communications Family PDUs.

B. IEEE DIS DATA STREAM PDUs

In this thesis the author has developed a set of data stream PDUs that are focused on radio communications. In the Radio Communication Family of the DIS specification there are sets of PDUs, which accurately define the specification for radio communication. Within this family there are the *TransmissionPDU*, *ReceiverPDU*, and *SignalPDU*.

C. DIS IMPLEMENTATION

The DIS standard is implemented in Java, primarily programmed by Don McGregor and Don Brutzman at Naval Postgraduate School as part of the Web3D Consortium's DIS-Java-VRML working group (http://www.wed3d.org). This work adds the Radio Communications Protocol Family to this ongoing effort, which deal with the parameterization of radio communications. A number of other PDUs are also employed, including the *EntityStatePDU* (ESPDU). The ESPDU contains the entity's EntityID, location, orientation, and velocity information. This signals visualization is implemented with the Radio Communication Family of PDUs, which include the *TransmitterPDU*, *ReceiverPDU*, *SignalPDU* and ESPDU for state information. Coding these three PDUs

provides the ability to send the communications status across the network and update the shared virtual battlefield residing on distributed participant computers.

The first step in implementing DIS is establishing the network interface to read and write via multicast. The java.net library is the network interface to the operating system and contains all of the networking methods needed. The program first opens a multicast socket with a port number and then calls the *joinGroup()* method, which joins the socket to the provided multicast group address as shown in Figure 4.1.

```
try {
   multicastAddress = InetAddress.getByName(MULTICASTIP);
   multicastSock = new MulticastSocket(APP_PORT);
   multicastSock.setTimeToLive(1);
   multicastSock.joinGroup(multicastAddress);
}
   catch(IOException ioe) {
   System.out.println("Establish multicast UDP port error: "
   + ioe.getMessage());
   return;
}
```

Figure 4.1: Instantiation of a Multicast socket using Java's built-in networking.

The underlying DIS implementation is an invisible set of code to the user, but remains straightforward from the programmers' perspective. *DatagramStreamBuffer*, *BehaviorStreamBuffer* and *EntityDispatcher* objects begins threaded operation and then will handle the network interface, instantiate with the multicast IP address and port, and converting the datagrams to (and from) application PDUs. The VRML visualization receives values from the current *EntityDispatcher*, which pulls incoming PDUs from the network and dispatches them to the correct entity; in this case, the correct antenna. Figure 4.2 and 4.3 show these relationships. Further detail can be found in the DIS-Java-VRML Javadoc and source-code examples.

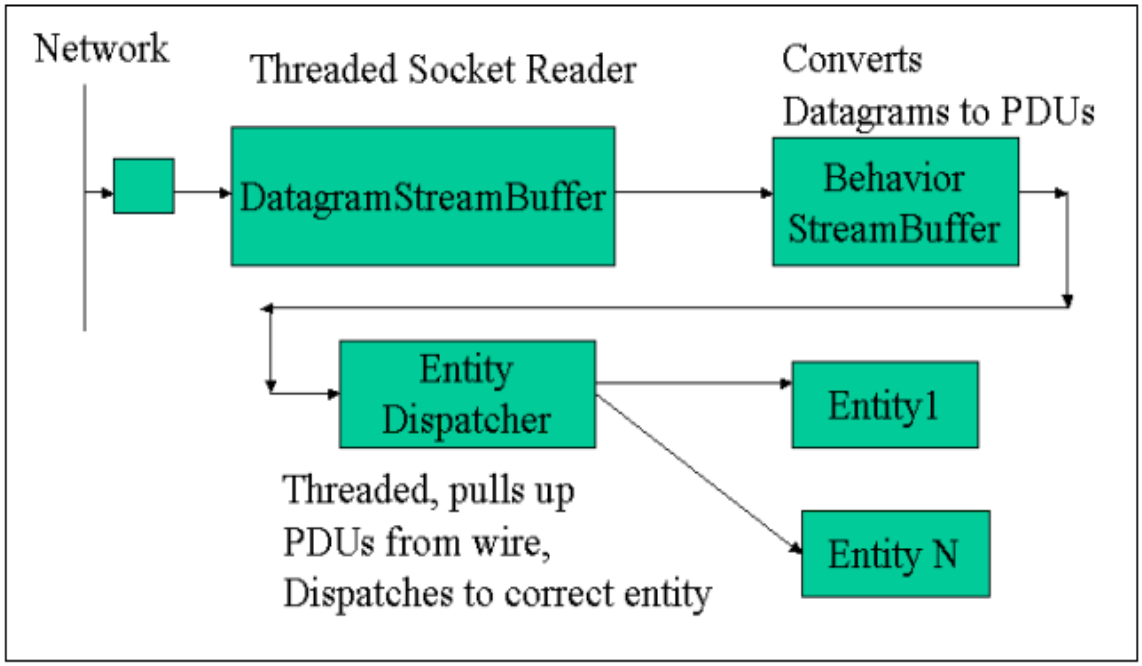


Figure 4.2. DIS networking paths to create entities in the visualization.

For signal visualization only the entity location field of the ESPDU was used. Since entity velocities are not implemented, the antenna simulators include no dead reckoning. Although this does not delay the 3D rendering process, when antenna updates are received the entity jumps to the new position. This may result in less believability for the visualization because the antenna is immediately repositioned to the changed value. Adding velocity information while repositioning antennas is a worthwhile entry-level task for future work.

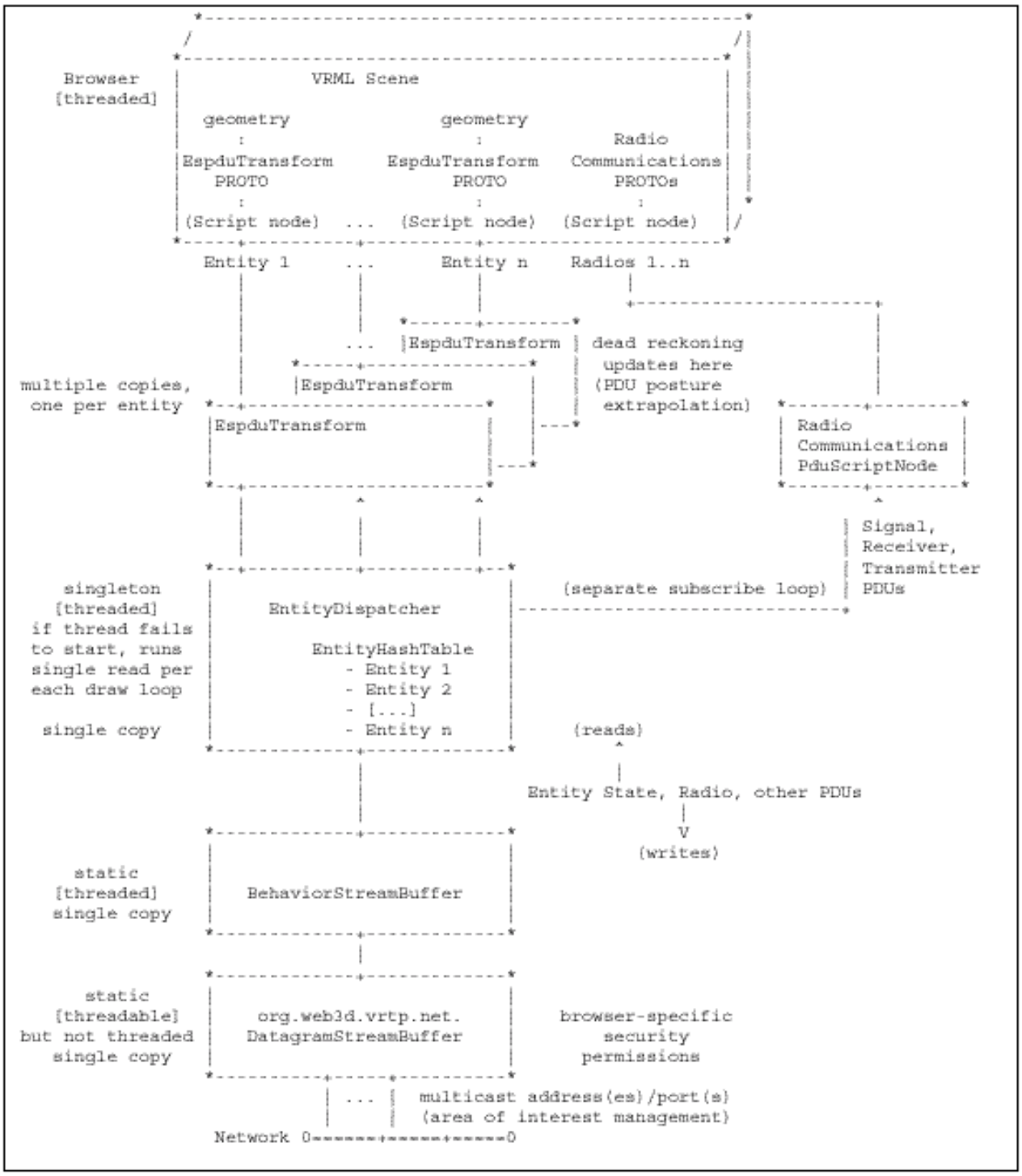


Figure 4.3: DIS-Java-VRML streaming stack, as documented in Javadoc from the *mil.navy.nps.dis.EntityDispatcher* package.

```
C:\vrtp\demo\helicopter>java demo.helicopter.StartPanel -pause 30
Pause before starting.............................
multicast timeToLive ttl=15
RTP headers prepended=false
entering actionPerformed() w/args...
   event = java.awt.event.ActionEvent[ACTION_PERFORMED,cmd=Antenna] on
button0
* * * Stand by start Antenna Panel (listen 10 seconds for other
players...)
 TEST jButton_OpenFile_actionPerformed

Antenna 5880, 10TH, 0, 0, 15.0, 183, SHF    Good Line Read
Antenna 58I01, 10TH, 17000, 17000, 15.0, 181, SHF    Good Line Read
Antenna 58I02, 10TH, -12000, 6000, 15.0, 183, SHF    Good Line Read
Antenna 58I03, 10TH, 4500, -8000, 15.0, 183, SHF    Good Line Read
Antenna 58I05, 10TH, 5000, 5000, 15.0, 198, SHF    Good Line Read
Antenna 58I06, 10TH, 22000, -26000, 15.0, 183, SHF    Good Line Read
Got strategy of org.web3d.vrtp.security.SunSecurityStrategy
org.web3d.vrtp.net.DatagramStreamBuffer: *** constructor new instance
multicast org.web3d.net.DatagramStreamBuffer
    (224.2.181.145, 62040)
    (this should only appear once..)
Got strategy of org.web3d.vrtp.security.SunSecurityStrategy
org.web3d.vrtp.net.DatagramStreamBuffer: strategy.invokePrivilege(this,
"createMulticastSocket", [224.2.181.145/224.2.18
1.145 62040]);
org.web3d.vrtp.net.DatagramStreamBuffer: joined multicast address
224.2.181.145, port 62040
org.web3d.vrtp.net.DatagramStreamBuffer: setDEBUG false
BehaviorStreamBuffer: commencing datagramStreamBuffer.run ()...
org.web3d.vrtp.net.DatagramStreamBuffer: run:
strategy.invokePrivilege(this, "doRun");
org.web3d.vrtp.net.DatagramStreamBuffer: commence doRun ();
Listening for other vehicles........BehaviorStreamBuffer: shutdown ();
```

Figure 4.4. : Text output of the start of the Antenna Visualization sending PDUs.

D. RADIO COMMUNICATIONS PROTOCOL (RCP) FAMILY

The PDUs of the Radio Communications protocol family are described in paragraphs 5.3.8.1 through 5.3.8.3 of the IEEE DIS specification. This family has the *TransmitterPDU*, *ReceiverPDU*, and the *SignalPDU*, now coded in Java and added to the DIS-Java-VRML code library. The PDU and attribute definitions are extracted and accepted in complete detail here. Implementers need to remain consistent with these

semantic definitions in order to maintain functional interoperability with other DIS systems.

1. Transmitter PDU

Transmitter PDU provides detailed information about a radio transmitter, which is communicated by issuing a Transmitter PDU. See Appendix A for implementation. The Transmitter PDU must contain the following fields:

- *PDU Header.* This field shall contain data common to all DIS PDUs.
- *Entity ID.* This field shall identify the entity that is the source of the radio transmission.
- *Radio ID.* This field shall identify a particular radio within a given entity. Radio IDs shall be assigned sequentially to the radios within an entity, starting with Radio ID 1. The combination of Entity ID and Radio ID uniquely identify a radio within a simulation exercise.
- *Radio Entity Type.* This field shall indicate the type of radio being simulated.
- *Transmit State.* This field shall specify whether a radio is off, powered but not transmitting, or powered and transmitting
- *Input Source.* This field shall specify which position, (pilot, co-pilot, gunnery officered.) or data port in the entity utilizing the radio, is providing the input audio or data being transmitted.
- *Antenna Location.* This field shall specify the location of the radiating portion of the antenna.
- *Antenna Pattern Type.* This field shall specify the type of representation used for the radiation pattern from the antenna.
- *Antenna Pattern Length.* This field shall specify the length in octets of the Antenna Pattern Parameters field.
- *Frequency.* This field shall specify the center frequency being used by the radio for transmission. This frequency shall be expressed in units of hertz.
- *Transmit Frequency Bandwidth.* This field shall identify the bandpass of the radio defined by the Radio ID field and the Radio Type field,
- *Power.* This field shall specify the average power being transmitted in units of decibel-milliwatts.
- *Modulation Type.* This field shall specify the type of modulation used for radio transmission.
- *Crypto System.* This field shall identify the crypto equipment utilized if such equipment is used with the Transmitter PDU.

- *Crypto Key ID.* This field, shall indicate that the crypto equipment is in the baseband encryption mode, and when set shall indicate that the crypto equipment is in the diphase encryption mode.
- *Length of Modulation Parameters.* These fields shall specify the length in octets of the modulation parameters.

Field size (bits)	Transmitter 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
		Host—16-bit unsigned integer
		Entity—16-bit unsigned integer
16	Radio ID	16-bit unsigned integer
64	Radio Entity Type	Entity Kind—8-bit enumeration
		Domain—8-bit enumeration
		Country—16-bit enumeration
		Category—8-bit enumeration
		Nomenclature Version—8-bit enumeration
		Nomenclature—16-bit enumeration
8	Transmit State	8-bit enumeration
8	Input Source	8-bit enumeration
16	Padding	16 bits unused
192	Antenna Location	*X*-component—64-bit floating point
		Y-component—64-bit floating point
		Z-component—64-bit floating point
96	Relative Antenna Location	*x*-component—32-bit floating point
		y-component—32-bit floating point
		z-component—32-bit floating point
16	Antenna Pattern Type	16-bit enumeration
16	Antenna Pattern Length (*L*)	16-bit unsigned integer
64	Frequency	64-bit unsigned integer
32	Transmit Frequency Bandwidth	32-bit floating point

(Table continued on next page)

Field size (bits)	Transmitter PDU fields	
32	Power	32-bit floating point
64	Modulation Type	Spread spectrum—16-bit Boolean array
		Major—16-bit enumeration
		Detail—16-bit enumeration
		System—16-bit enumeration
16	Crypto System	16-bit enumeration
16	Crypto Key ID	16-bit unsigned integer
8	Length of Modulation Parameters (l)	8-bit unsigned integer
24	Padding	24 bits unused
Modulation Parameter # 1		
⋮		
Modulation Parameter # N		
Antenna Pattern Parameter #1		
⋮		
Antenna Pattern Parameter #M		
Total Transmitter PDU size $=832 + 8l + 8L$ where l is length of modulation parameters L is antenna pattern length		

Figure 4.5: Transmitter PDU Diagram From Ref (DIS IEEE Std 1278.1a-1995).

```
C:\vrtp>java mil.navy.nps.testing.TransmitterPduSender
Got strategy of org.web3d.vrtp.security.SunSecurityStrategy
org.web3d.vrtp.net.DatagramStreamBuffer: *** constructor new instance
multicast org.web3d.net.DatagramStreamBu
ffer
    (224.2.181.145, 62040)
    (this should only appear once..)
Got strategy of org.web3d.vrtp.security.SunSecurityStrategy
org.web3d.vrtp.net.DatagramStreamBuffer: strategy.invokePrivilege(this,
"createMulticastSocket", [224.2.181.14
5/224.2.181.145 62040]);
org.web3d.vrtp.net.DatagramStreamBuffer: joined multicast address
224.2.181.145, port 62040
org.web3d.vrtp.net.DatagramStreamBuffer: setDEBUG false

 Sending the Transmitter PDU


C:\vrtp>
```

Figure 4.6: Text output of test program *mil.navy.nps.testing.TransmitterPduSender* showing sending of a Transmitter PDU.

```
C:\vrtp>java mil.navy.nps.testing.PduTestListener
Got strategy of org.web3d.vrtp.security.SunSecurityStrategy
org.web3d.vrtp.net.DatagramStreamBuffer: *** constructor new instance multicast
org.web3d.net.DatagramStreamBu
ffer
    (224.2.181.145, 62040)
    (this should only appear once..)
Got strategy of org.web3d.vrtp.security.SunSecurityStrategy
org.web3d.vrtp.net.DatagramStreamBuffer: strategy.invokePrivilege(this,
"createMulticastSocket", [224.2.181.14 5/224.2.181.145 62040]);
org.web3d.vrtp.net.DatagramStreamBuffer: joined multicast address 224.2.181.145,
port 62040
org.web3d.vrtp.net.DatagramStreamBuffer: setDEBUG false
BehaviorStreamBuffer: commencing datagramStreamBuffer.run ()...
org.web3d.vrtp.net.DatagramStreamBuffer: run:  strategy.invokePrivilege(this,
"doRun");
org.web3d.vrtp.net.DatagramStreamBuffer: commence doRun ();
Testing the get methods
In PduTestListener, received a  TransmitterPdu, EntityID = (site, application,
entity ID) = (32, 33, 7)
In PduTestListener, received a  TransmitterPdu, RadioID = 0
In PduTestListener, received a  TransmitterPdu, RadioEntityType =  entityKind = 0
domain = 0 country= 0 catego
ry =0 nomenclatureVersion = 0 padding1 = 0
In PduTestListener, received a  TransmitterPdu, TransmitState = 3
In PduTestListener, received a  TransmitterPdu, InputSource = 2
In PduTestListener, received a  TransmitterPdu, AntennaLocation =
mil.navy.nps.dis.WorldCoordinate@d107f
In PduTestListener, received a  TransmitterPdu, elativeAntennaLocation =
mil.navy.nps.dis.EntityCoordinate@360 be0
In PduTestListener, received a  TransmitterPdu, AntennaPatternType = 44
In PduTestListener, received a  TransmitterPdu, AntennaPatternLength = 10
In PduTestListener, received a  TransmitterPdu, Frequency = 100
In PduTestListener, received a  TransmitterPdu, TransmitFrequencyBandwidth = 330
In PduTestListener, received a  TransmitterPdu, Power = 555
In PduTestListener, received a  TransmitterPdu, ModulationType  =
 SpreadSpectrum = 0 Major = 0 Detail = 0 sSystem = 0

In PduTestListener, received a  TransmitterPdu, CryptoSytem = 30
In PduTestListener, received a  TransmitterPdu, CryptoKeyId = 40
In PduTestListener, received a  TransmitterPdu, LengthOfModulationParameters = 12
Test the toString Method
In PduTestListener, received a  TransmitterPdu,  Transmitter Object.  =
EntityID = (site, application, entity ID) = (32, 33, 7)
RadioID = 0
RadioEntityType =  entityKind = 0 domain = 0 country= 0 category =0
nomenclatureVersion = 0 padding1 = 0
TransmitState = 3
InputSource = 2
Padding1 = 0
AntennaLocaction = mil.navy.nps.dis.WorldCoordinate@26b249
RelativeAntennaLocation = mil.navy.nps.dis.EntityCoordinate@2f0db
AntennaPatternType = 44
AntennaPatternLength = 10
Frequency = 100
TransmitFrequencyBandwidth = 330
Power = 555
```

Figure 4.7: Text output of test program *mil.navy.nps.testing.PduTestListener* showing proper receipt of a Transmitter PDU.

2. Receiver PDU

The Receiver PDU provides detailed information about the communication an emitter's receiver state shall be communicated with a Receiver PDU. See Appendix A for implementation. The Receiver PDU must contain the following fields:

- *PDU Header.* This field shall contain data common to all DIS PDUs.
- *Entity ID.* This field shall identify the entity that is controlling the radio transmission. The source entity may either represent the radio itself or represent an entity (such as a vehicle) that contains the radio.
- *Radio ID.* This field shall identify a particular radio within a given entity. Radio IDs shall be assigned sequentially to the radios within an entity, starting with Radio ID 1. The combination of Entity ID and Radio ID uniquely identifies a radio within a simulation exercise.
- *Receiver State.* This field shall indicate the state of the receiver, which shall either be idle or active.
- *Received Power.* This field shall indicate the radio frequency power received, after applying any propagation loss and antenna gain.
- *Transmitter Entity ID.* This field shall identify the entity that is the source of the transmission that is currently being received. The selection of the received transmitter depends on the characteristics and state of the simulated receiver.
- *Transmitter Radio ID.* This field shall identify the particular radio within the entity cited in item f that is the source of the radio transmission.

Field size (bits)	Receiver 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
16	Radio ID	16-bit unsigned integer
16	Receiver State	16-bit enumeration
16	Padding	16 bits unused
32	Received Power	32-bit floating point
48	Transmitter Entity ID	Site—16-bit unsigned integer
		Application—16-bit unsigned integer
		Entity—16-bit unsigned integer
16	Transmitter Radio ID	16-bit unsigned integer
Total Receiver PDU size = 288 bits		

Figure 4-8: Receiver PDU Diagram From Ref (DIS IEEE Std 1278.1a-1995).

```
C:\vrtp>java mil.navy.nps.testing.ReceiverPduSender
Got strategy of org.web3d.vrtp.security.SunSecurityStrategy
org.web3d.vrtp.net.DatagramStreamBuffer: *** constructor new instance
multicast org.web3d.net.DatagramStreamBu
ffer
    (224.2.181.145, 62040)
    (this should only appear once..)
Got strategy of org.web3d.vrtp.security.SunSecurityStrategy
org.web3d.vrtp.net.DatagramStreamBuffer: strategy.invokePrivilege(this,
"createMulticastSocket", [224.2.181.14
5/224.2.181.145 62040]);
org.web3d.vrtp.net.DatagramStreamBuffer: joined multicast address
224.2.181.145, port 62040
org.web3d.vrtp.net.DatagramStreamBuffer: setDEBUG false

 Sending the Receiver PDU


C:\vrtp>
```

Figure 4.9: Text output of test program *mil.navy.nps.testing.ReceiverPduSender* showing sending of a Receiver PDU.

```
C:\vrtp>java mil.navy.nps.testing.PduTestListener
Got strategy of org.web3d.vrtp.security.SunSecurityStrategy
org.web3d.vrtp.net.DatagramStreamBuffer: *** constructor new instance
multicast org.web3d.net.DatagramStreamBu
ffer
    (224.2.181.145, 62040)
    (this should only appear once..)
Got strategy of org.web3d.vrtp.security.SunSecurityStrategy
org.web3d.vrtp.net.DatagramStreamBuffer: strategy.invokePrivilege(this,
"createMulticastSocket", [224.2.181.14
5/224.2.181.145 62040]);
org.web3d.vrtp.net.DatagramStreamBuffer: joined multicast address
224.2.181.145, port 62040
org.web3d.vrtp.net.DatagramStreamBuffer: setDEBUG false
BehaviorStreamBuffer: commencing datagramStreamBuffer.run ()...
org.web3d.vrtp.net.DatagramStreamBuffer: run:
strategy.invokePrivilege(this, "doRun");
org.web3d.vrtp.net.DatagramStreamBuffer: commence doRun ();

Testing the get methods

In PduTestListener, received a Receiver PDU, EntityID = (site,
application, entity ID) = (99, 20, 8)
In PduTestListener, received a Receiver PDU, RadioID = 77
In PduTestListener, received a Receiver PDU, ReceiverState = 50
In PduTestListener, received a Receiver PDU, ReceiverPower = 8.3
In PduTestListener, received a Receiver PDU, TransmitterEntityID =
(site, application, entity ID) = (0, 0, 0)
In PduTestListener, received a Receiver PDU, TransmitterRadioID = 222

Test the toString Method

In PduTestListener, received a  ReceiverPDU,  ReceiverObject.  =
EntityID = (site, application, entity ID) = (99, 20, 8)
RadioID = 77
ReceiverState = 50
Padding = 0
ReceiverPower = 8.3
TransmitterEntityID = (site, application, entity ID) = (0, 0, 0)
TransmitterRadioID = 222
```

Figure 4.10: Text output of test program *mil.navy.nps.testing.PduTestListener* showing proper receipt of a Receiver PDU.

3. Signal PDU

The Signal PDU is the actual transmission of voice audio or other radio emissions data must be communicated by issuing a Signal PDU. See Appendix A for implementation. The Signal PDU must contain the following fields:

- *PDU Header.* This field shall contain data common to all DIS PDUs.
- *Entity ID.* This field shall identify the entity that is the source of the radio transmission. The source entity may either represent the radio itself or represent an entity (such as a vehicle) that contains the radio.
- *Radio ID.* This field shall identify a particular radio within a given entity. The Entity ID, Radio ID pair associates each Signal PDU with the preceding Transmitter PDU that contains the same Entity ID, Radio ID pair. The combination of Entity ID and Radio ID uniquely identifies a particular radio within a simulation exercise.
- *Encoding Scheme.* This field shall specify the encoding used in the Data field of this PDU. The sample rate is in samples per second for audio data. The bit rate is in bits per second for digital data. The interpretation of the Data field of the Signal PDU shall depend on the value of encoding class.
- *TDL Type.* This field shall specify the TDL Type as a 16-bit enumeration field when the encoding class is the raw binary, audio, application-specific, or database index representation of a TDL message. When the Data field is not representing a TDL Message, this field shall be set to zero.
- *Sample Rate.* This field shall specify either the sample rate in samples per second if the encoding class is encoded audio or, the data rate in bits per second for data transmissions. If the encoding class is database index, this field shall be zero.
- *Data Length.* This field shall specify the number of bits of digital voice audio or digital data being sent in this Signal PDU.
- *Samples.* This field shall specify the number of samples in this PDU.
- *Data.* This field shall specify the audio or digital data conveyed by the radio transmission. If the encoding class is *encoded audi*o, the Data field shall be interpreted as containing audio information digitally encoded as specified by the encoding type. If the encoding class is *raw binary dat*a.

Field Size (bits)	Signal 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
16	Radio ID	16-bit unsigned integer
16	Encoding Scheme	16-bit enumeration
16	TDL Type	16-bit enumeration
32	Sample Rate	32-bit integer
16	Data Length	16-bit integer
16	Samples	16-bit integer
8	Data #0	8-bit unsigned integer
	⋮	
8	Data #*N*	8-bit unsigned integer
Total Signal PDU size = 256 + Data Length + 0 to 31 padding bits to increase the total Signal PDU size to a multiple of 32 bits.		

Figure 4.11: Signal PDU Diagram From Ref (DIS IEEE Std 1278.1a-1995).

```
C:\vrtp>java mil.navy.nps.testing.SignalPduSender
Got strategy of org.web3d.vrtp.security.SunSecurityStrategy
org.web3d.vrtp.net.DatagramStreamBuffer: *** constructor new instance
multicast org.web3d.net.DatagramStreamBu
ffer
    (224.2.181.145, 62040)
    (this should only appear once..)
Got strategy of org.web3d.vrtp.security.SunSecurityStrategy
org.web3d.vrtp.net.DatagramStreamBuffer: strategy.invokePrivilege(this,
"createMulticastSocket", [224.2.181.14
5/224.2.181.145 62040]);
org.web3d.vrtp.net.DatagramStreamBuffer: joined multicast address
224.2.181.145, port 62040
org.web3d.vrtp.net.DatagramStreamBuffer: setDEBUG false

Sending the Signal PDU


C:\vrtp>
```

Figure 4.12: Text output of test program *mil.navy.nps.testing.SignalPduSender* showing sending of a Signal PDU.

```
C:\vrtp>java mil.navy.nps.testing.PduTestListener
Got strategy of org.web3d.vrtp.security.SunSecurityStrategy
org.web3d.vrtp.net.DatagramStreamBuffer: *** constructor new instance
multicast org.web3d.net.DatagramStreamBu
ffer
    (224.2.181.145, 62040)
    (this should only appear once..)
Got strategy of org.web3d.vrtp.security.SunSecurityStrategy
org.web3d.vrtp.net.DatagramStreamBuffer: strategy.invokePrivilege(this,
"createMulticastSocket", [224.2.181.14
5/224.2.181.145 62040]);
org.web3d.vrtp.net.DatagramStreamBuffer: joined multicast address
224.2.181.145, port 62040
org.web3d.vrtp.net.DatagramStreamBuffer: setDEBUG false
BehaviorStreamBuffer: commencing datagramStreamBuffer.run ()...
org.web3d.vrtp.net.DatagramStreamBuffer: run:
strategy.invokePrivilege(this, "doRun");
org.web3d.vrtp.net.DatagramStreamBuffer: commence doRun ();

Testing the get methods

In PduTestListener, received a SignalPdu, EentityID = (site,
application, entity ID) = (14, 17, 9)
In PduTestListener, received a SignalPdu, RadioID = 77
In PduTestListener, received a SignalPdu, EncodingScheme = 50
In PduTestListener, received a SignalPdu, TDLType = 55
In PduTestListener, received a SignalPdu, SampleRate = 888
In PduTestListener, received a SignalPdu, DataLength = 357
In PduTestListener, received a SignalPdu, Samples = 411
In PduTestListener, received a SignalPdu, Data0 = 4
In PduTestListener, received a SignalPdu, Data0 = 4
In PduTestListener, received a SignalPdu, Data0 = 4
In PduTestListener, received a SignalPdu, Data0 = 4
In PduTestListener, received a SignalPdu, Data0 = 4
In PduTestListener, received a SignalPdu, Data0 = 4
In PduTestListener, received a SignalPdu, Data0 = 4
In PduTestListener, received a SignalPdu, Data0 = 4
In PduTestListener, received a SignalPdu, Data0 = 4
In PduTestListener, received a SignalPdu, Data0 = 4
In PduTestListener, received a SignalPdu, Data0 = 4

Test the toString Method

In PduTestListener, received a  SignalPdu,  SignalObject.  =
EntityID = (site, application, entity ID) = (14, 17, 9)
RadioID = 77
EncodingScheme = 50
TdlType = 55
SampleRate = 888
DataLength = 357
Samples = 411
data = 4 4 4 4 4 4 4 4 4 4 4
```

Figure 4.13: Text output of test program *mil.navy.nps.testing.PduTestListener* showing proper receipt of a Signal PDU.

E. SUMMARY

This chapter presents many of the practical factors involved in implementing the DIS protocol in this visualization. Using the DIS protocol allows this visualization to have a common agreed-upon format which other computers on the network can understand. When the visualization is shared among multiple participants, each one can view a common picture of the signals overlaid on the same virtual battlefield.

V. TACTICAL VISUALIZATION OF BATTLEFIELD EMISSIONS

A. INTRODUCTION

This chapter explores a 3D visualization model designed to present the results of the DIS-Java-VRML simulation. The chapter presents an overview of signals visualization, tactical visualization and visualization in general. It also addresses the issues of dimensional space, pertinent 3D graphics techniques, and how the visualization parameters to correspond MSE-NPT data files. The chapter concludes with visualization recommendations for tactical visualization.

B. OVERVIEW

Simulations typically require some form of visualization. Whether the visualization is in the form of text tables or 2D graphic plots, the simulation must produce data in a logical and orderly grouping of similar information. Before the advent of computer data collection, information was collected in tables and occasionally plotted on paper or acetate, usually by hand. With the advent of computer technology, the mainstay of data visualization shifted from text tables to 2D graphic plots as seen in Figure 5.1, usually prepared manually using a presentation or graphics package. With the increase in computational power of computers and the widespread availability of 3D rendering software, both for the web and stand-alone use, it is possible to undertake paradigm shift in visualization using 3D rendering.

In the Army of today, tactical signal planning is still performed from radio-profile tables, 2D fresnel plots and transmission range fans. Such an approach was sufficient twenty years ago when less signal data was processed and evaluated. Today the sheer volume of planning data from radio frequencies, antenna polarization, and radio ranges (as well as the overlay of the data communications network information) demands a more robust visualization of the incoming data. The logical next step in signals planning is the addition of third dimension to these visualizations. Margarida Karahalios discusses the history of data visualization in her thesis titled, "Underwater Source Localization Using Scientific Data Visualization," in which she states that half of the human neo-cortex is devoted to visual information processing. Since the human brain is wired for stereoscopic visual data input, which is typically 3D in nature, the natural conclusion continues to be 3D representations.

C. VISUALIZATION CONSIDERATIONS

There are many visualization considerations that should be taken into account when developing a system of this type. This section addressed the areas of tactical visualization, dimensional space, available graphics technique and parameters mapping MSE-NPT data to graphics parameters, and some initial visualization recommendations.

1. Tactical Visualization

In the U.S. Army today, visualization of four-dimensional space-time is generated through the use of 2D representations. These representations have various combinations of unit operational graphics, based on the position and time sequence with information

presented by phase of operation. Threat overlay, logistical overlay, communications planning overlay, and operational strategy are examples of representations, in widespread use. Typically these 2D representations are evaluated in real time by showing the situational picture of the current battle space. This is often performed on separate computers displays at the same time, with several computer operators and one supervisor interpreting the 2D representations. The supervisor's mission is to mentally integrate the representations and develop a four-dimensional (space-time) mental model of the current tactical picture in the real world. The signal planner must also see how the tactical picture fits together with the signal support plan to ensure the commander has the communications coverage he needs.

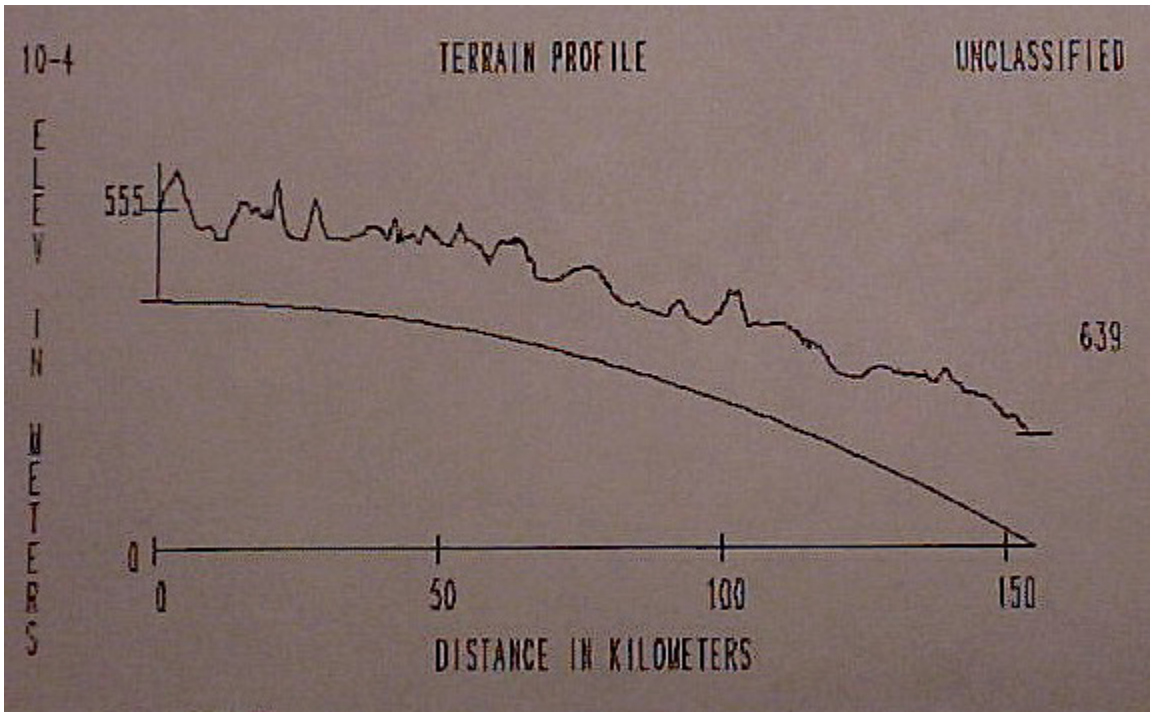


Figure 5.1: MSE-NPT Terrain Profile From Ref (TB 11-5895-1544-10-1).

In years past, when voice and data message volume was much lighter and fewer communication shelters were deployed, planning requirements for the signal planner were not crushing. Today, increased voice and data traffic have dramatically increased the volume of information processed by the signal planner along with the time needed to plan mission and responsibilities. This frequently leads to an information overload for the planner. It is extremely difficult for an information-overloaded supervisor to make accurate decisions while mentally integrating several overlays into a multiple-dimensional overlay representation.

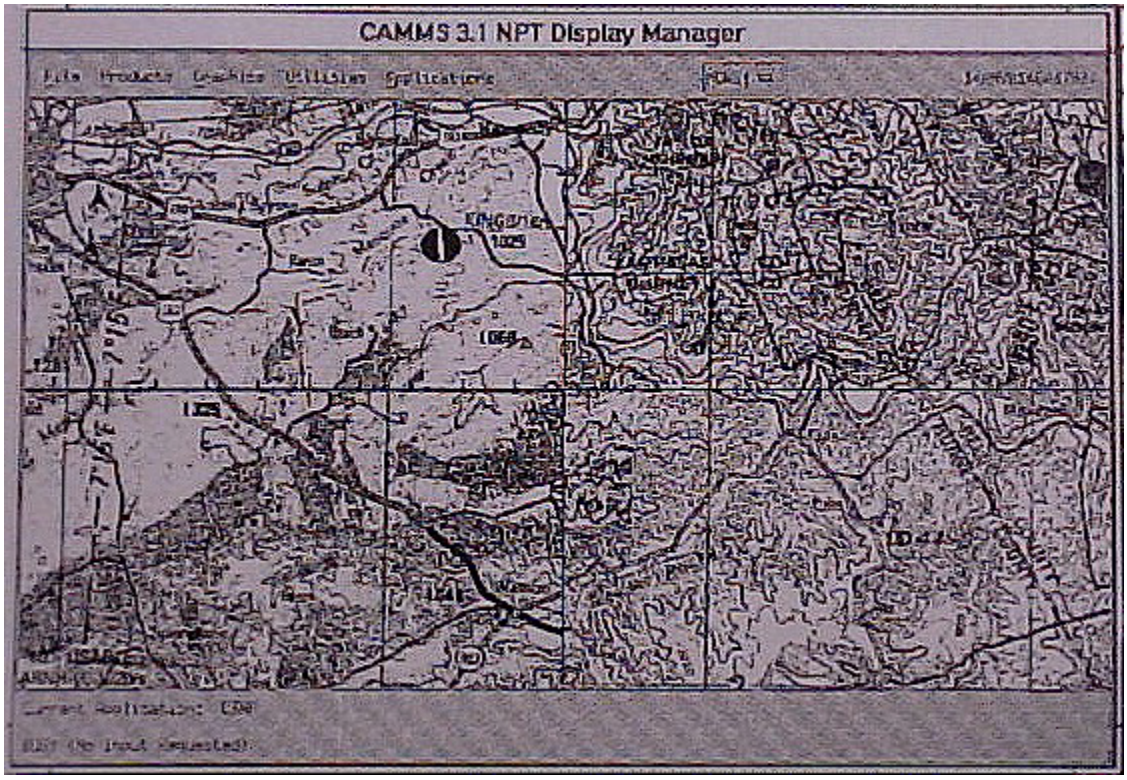


Figure 5.2: MSE-NPT Terrain Screen Shot of DTED Map
From Ref (TB 11-5895-1544-10-1).

In the Army, visualization of radio-wave propagation has not progressed much past the 2D state of the art, as shown in Figure 5.1 and 5.2. Typically the extent of visualization is colorful 2D plots. Nevertheless, when complex communication overlays are being investigated, ingenious use of 3D representations with color can provide the signal planner with a higher order of data-variable representation. Producing a simulated four and five-dimensional profile representation. Like most physically based solutions, judicious simplification and abstraction are a must. Scientific visualization of radio-wave propagation data will also benefit from careful simplification and abstraction.

2. Dimensional Space

Radio-wave propagation data is typically a volumetric data set covering an area on and above the ground, as opposed to a surface data set, which strictly covers the ground level. In addition, wave propagation data has various parameters of interest that can consider as additional dimensions. Radio-frequency intensity level, bit-error rate (BER), and signal-to-noise ratio are three examples of these extra dimensions. In the visualization discipline, such a problem domain is termed a “high-dimensional data space.” For the examples mentioned above, 3D or 4D representations might be generated to simultaneously render all relevant planning parameters. How can such representations be generated? In 2D plotting, the techniques of coloring and use of contours can transform a representation into a pseudo three-dimensional representation. Similar techniques can be applied in 3D renderings. By mapping additional dimensions to graphics quantities besides spatial, pseudo high-dimensional space plots can be generated (Keller 1993).

3. Available Graphics Techniques and Parameters

A number of 3D techniques must be considered when mapping multiple dimensions of data to a visual representation. When choosing the most appropriate visualization technique, visualization designers must consider what the parameters of the data are. In general term the signal planner needs to represent the propagation data over the terrain with respect to the movement of forces and operational flow of the battle space. In general terms visualization technique needs to take into consideration the seven major areas of information visualization as shown in Figure 5.3.

1. **Comparing**: images, positions, data set, sub sets of data
2. **Distinguishing**: importance objects activities, ranges of value
3. **Indicating direction**: orientation, order direction of flow
4. **Locating**: position relative to axis, object map
5. **Relating**: concepts with regards to, value and direction, position and shape, temperature and velocity, object type and value.
6. **Representing values**: numeric value of data
7. **Revealing objects**: exposing, highlighting, bring to the front, making visible, enhancing visibility.

Figure 5.3: Seven major tasks for design of information visualization (Keller 1993).

The signal planner is concerned with all the above areas. The signal planner needs to compare and distinguish wave propagation overlays for frequency de-confliction. The signal planner also needs to indicate the direction and location of the antenna emitter, and particularly the direction of antennas to be masked from enemy detection. By providing the commander with a representation of the signal plan

connected to the operational movement plan, signal planners can more easily recognize any potential problems. Representing the power output of the antennas as graphic primitives simply exposes a communications problem in a non-technical fashion. For this reason, when one decides to implement high-dimensional space-data rendering, careful attention must be given to ensuring that the visual parameters employed do not overlap in such a way that an ambiguous or incomprehensible data visualization results. Such fundamental challenges form the basis of scientific visualization research.

4. Mapping MSE-NPT Data to Graphics Parameters

The task of mapping MSE-NPT data must be addressed carefully. Support for multiple types of MSE-NPT file layouts is problematic, due to the number of way in which they can be formatted. Also the development of a set of geometric representations for the antennas (as well as the corresponding wave formations for the emitters), is challenging. If the wave formation does not approximate the true wave output from the emitter, such misrepresentations can cause interpretation problems when attempting to understand the emitter's true capabilities. The power output intensity of the wave, antenna dish azimuth, and antenna dish elevation must also be taken into consideration. By using clear VRML graphic representations in conjunction with the corresponding DIS PDU types for transmitters, receivers, and signals, a hierarchy of physical and logical parameters can be developed. In other words as the MSE-NPT file is parsed in, each field corresponds to a part of the 3D scene. This one-to-one correspondence of parameters to graphic representations ensures the most accurate representation of the MSE-NPT data. Examples of physical parameters include the location and direction of

the antennas. An example of a logical parameter is colorization of wave intensities in combination with textual information provides the user, with any additional information that could not be rendered as a geometric object in the scene. The hierarchy of parameters must be selected in a way that the parameters on the lower end of the hierarchy have little influence on the parameters on the higher end. Careful design of this type of hierarchy may prevent the rendering engine from eliminating important data from the visualization.

5. Initial Visualization Recommendations

The initial recommendation for visualizing MSE-NPT data is that three spatial components are mapped onto three spatial coordinates of the visualization system. This one-to-one correspondence forms the coordinate field and the elevation field as shown in Figure 5.4

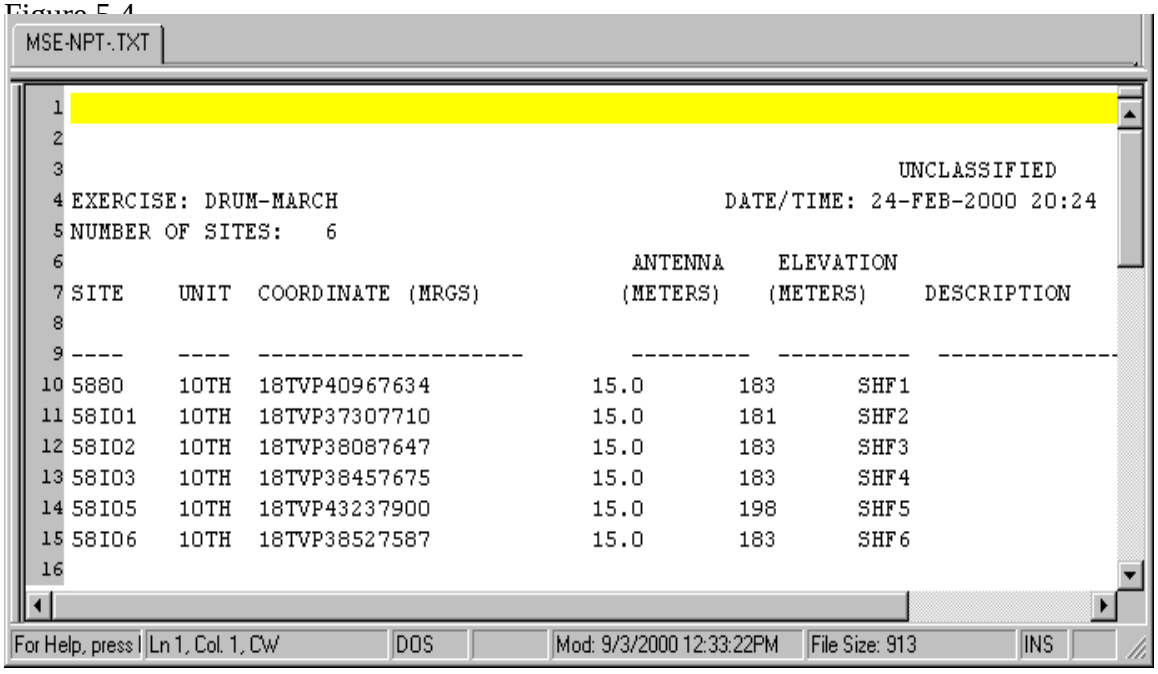


Figure 5.4: MSE-NPT antenna data for coordinates, height and elevation are mapped directly to 3D rendering coordinates.

For this initial test of the DIS-Java-VRML visualization, the MRGS coordinates were manually converted into their corresponding VRML coordinates. The updated MSE-NPT file shown in Figure 5.5 contains the VRML coordinates. The description field is mapped to appropriate VRML geometry, which is loaded into the visualization via a PROTOINSTANCE fieldValue update in the MSE-NPT file.

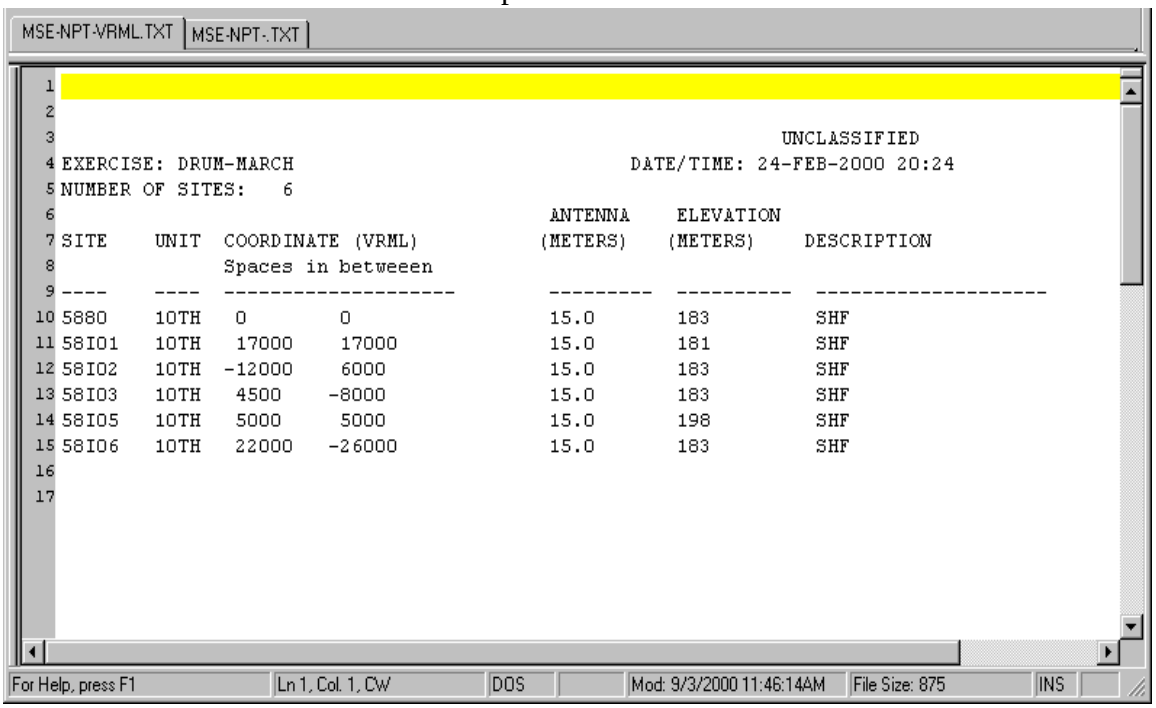


Figure 5.5: MSE-NPT file with VRML coordinates converted from MGRS coordinates.

The antenna power out, azimuth and elevation values are loaded from the corresponding MSE-NPT file into the *TransmitterPDU*, *ReceiverPDU* and *SignalPDU*. These values are rendered in the visualization as an directional transparent geodesic dome and a transparent dynamic beam, each with an associated text box enabling quantitative interpretation in a 3D context. The antenna power out is mapped to color so as the signal planner moves around the scene he can easily identify if receivers are within a coverage area or not. These initial visualization recommendations are provided as a starting point

for evaluation of the most important planning parameters provided by the MSE-NPT File. More work is needed to finish visualizing the full set of analytic data.

D. SUMMARY

With current communications planning requirements continuing to accelerate, innovations from signals-visualization can ease the information burden placed on the field planner evaluating the operational requirements. Creative and ingenious ways of displaying 3D data must be devised if higher information density is to be processed by a human. This new level of abstraction will provide enhanced understanding of the problem under consideration, and significantly enhance one's ability to make more rapid decisions.

VI. SYSTEMS INTEGRATION AND DEMONSTRATION

A. INTRODUCTION

This chapter discusses the development of the DIS-Java-VRML visualization of the MSE-NPT for signal planners. It also provides a step-by-step demonstration of how to run the DIS-Java-VRML visualization.

B. DEMONSTRATION

This section steps through the procedures for the setup of the MSE-NPT visualization. After the user installs the DIS-Java-VRML distribution, (http://www.web3d.org/WorkingGroups/vrtp/dis-java-vrml), four icons are added to desktop for easy access into the visualization. A complete set of the installation instructions appears on the "download" page in each DIS-Java-VRML distribution.



Figure 6.1: DIS-Java-VRML desktop icons.

Each icon has a separate function. These icons are linked to batch files, which can be edited for ease of system reconfiguration. The "Player Select" icon allows the user to select the control panel for an entity in which they are interested. The "Play Capture the Flag" icon is the central starting point for the DIS-Java-VRML framework

and it initiates, the player selection, the VRML world and the referee. The "Referee" is a Java-based application, which provides the arbitration between the entities in the tank and helicopter simulation regarding the flag capture and release. There are currently human controlled and agent controlled tank and helicopter simulations, single and team humanoid controlled simulations, and the MSE-NPT antenna visualization.

After the user starts the visualization by clicking the "Play Capture the Flag," icon the start panel appears as shown in Figure 6.2. This is the central selection menu for the game. This panel was easily configured to include the Antenna panel. Since the core programming language is Java, all developers need to have some proficiency in Java to make such changes.



Figure 6.2: DIS-Java-VRML Start Panel for Capture the Flag entities.

After a selection has been made on the start panel, the user will see two console windows on the desktop. One window displays the status of the Referee, and the other window corresponds to the control button being pressed by the user. The referee serves a vital purpose within the Capture the Flag game since it is the arbitrator between all entities in the visualization. The referee monitors the position, ID and status of each entity. If there is a conflict between two different entities regarding the flag capture, it is the function of the referee to arbitrate who is in control. This loading and arbitration is shown in Figure 6.3.

```
C:\vrtp\demo\helicopter>java demo.helicopter.Referee -pause 40
Pause before starting.....................................
multicast timeToLive ttl=15
RTP headers prepended=false
Got strategy of org.web3d.vrtp.security.SunSecurityStrategy
org.web3d.vrtp.net.DatagramStreamBuffer: *** constructor new instance multicast
org.web3d.net.DatagramStreamBuffer
   (224.2.181.145, 62040)
   (this should only appear once..)
Got strategy of org.web3d.vrtp.security.SunSecurityStrategy
org.web3d.vrtp.net.DatagramStreamBuffer: strategy.invokePrivilege(this, "createMulticastSocket",
[224.2.181.145/224.2.1
1.145 62040]);
org.web3d.vrtp.net.DatagramStreamBuffer: joined multicast address 224.2.181.145, port 62040
org.web3d.vrtp.net.DatagramStreamBuffer: setDEBUG false
BehaviorStreamBuffer: commencing datagramStreamBuffer.run ()...
org.web3d.vrtp.net.DatagramStreamBuffer: run:  strategy.invokePrivilege(this, "doRun");
org.web3d.vrtp.net.DatagramStreamBuffer: commence doRun ();
Listening 10 seconds for other Referee.....
No other Referee heard, starting up!
found FortIrwinTerrain.wrl, loading...
sending CreateEntity PDU...
sending Comment PDU with URL...
send initial red Pdu.    send initial blue pdu.
red  keepAlive PDU sent  blue keepAlive PDU sent
red  keepAlive PDU sent  blue keepAlive PDU sent
red  keepAlive PDU sent  blue keepAlive PDU sent
```

Figure 6.3: Text output of DIS-Java-VRML Referee copied from console.

This is particularly useful in the antenna visualization, where each antenna has a unique entity ID and Radio ID. If another user were to load an MSE-NPT file with the same set of information, the referee would only load antennas with an ID that did not conflict with those already loaded in the system. In the antenna visualization, these entities are the antenna emitters.

The networked visualization must load Java class files from the hard drive, the user must allow their browser to load these class files. This security measure ensures that the user only loads class files that are recognized and authenticated. Figure 6.4 shows the

Netscape permissions dialog needed to load unsigned classes. The class files that are subsequently loaded provide the connectivity between the VRML Visualization and the DIS-Java based networking. Future work in DIS-Java-VRML will sign these classes.

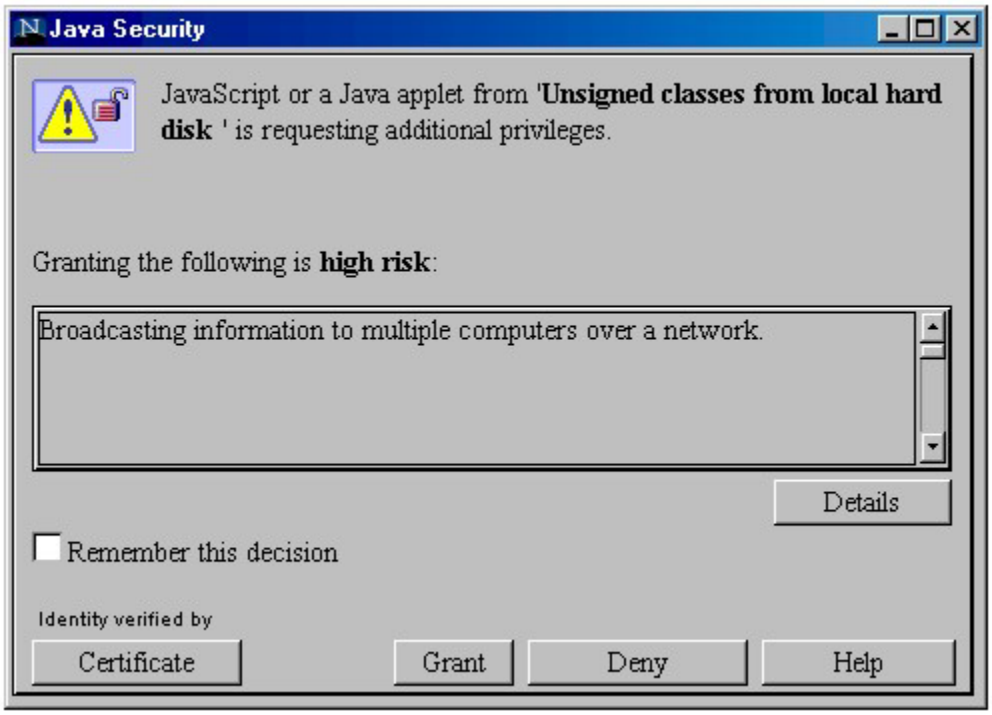


Figure 6.4: Netscape Java Security warning for reading unsigned class files.

After the "Antenna" button is pressed, the user is presented with the MSE-NPT File reader dialog box as seen in Figure 6.5. This is a separate Java-based application written to parse the MSE-NPT files into a format that can be understood by the DIS protocol. The user can then select from a number of options: read file, process and close, about, and exit. The user should choose to read in an MSE-NPT file. Once selected, the user will see a file dialog box seen in Figure 6.6.

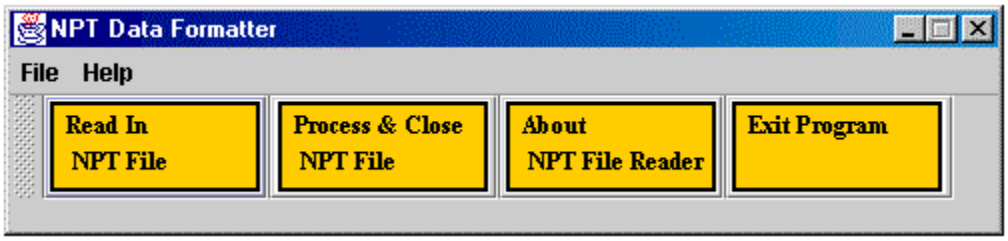


Figure 6.5: MSE-NPT loader selection panel.

After selecting the directory containing the MSE-NPT file, as seen in Figure 6.7, the user opens it. This loads the file into the computer memory. Next, the user must process the file.

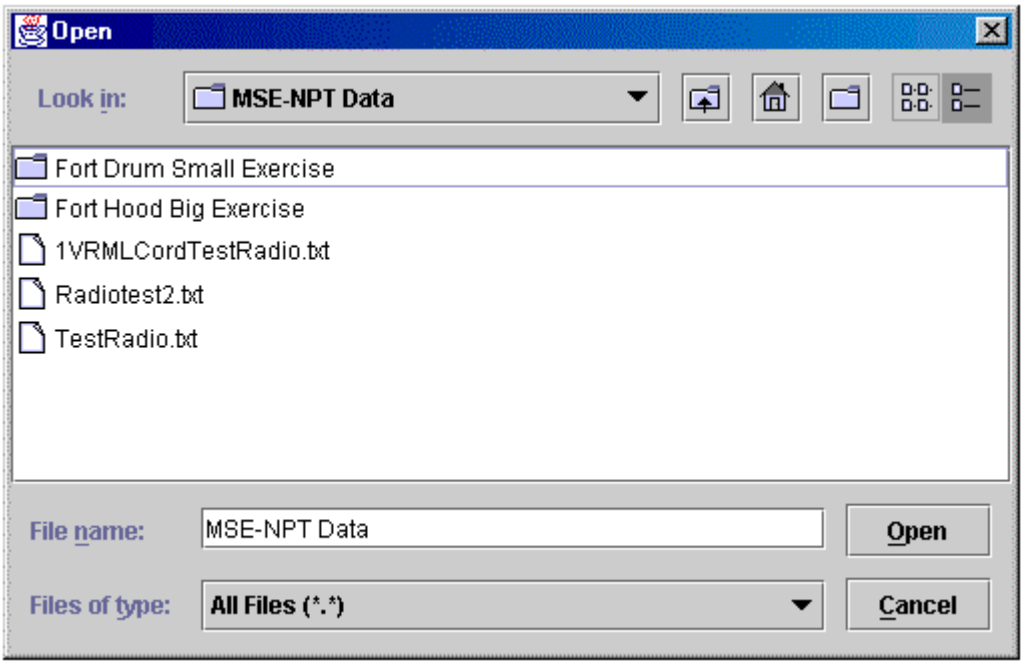


Figure 6.6: MSE-NPT file loader dialog box.

Once the user selects the “process and close” button, the MSE-NPT file reader process, the MSE-NPT file reader reads the file line by line parses it into Java antenna objects and then closes the data file. These objects are then loaded into the start panel, as shown in

Figure 6.8. There are two other buttons on the panel. The “About” button gives a brief description of the program and the “Exit” button terminates the application.

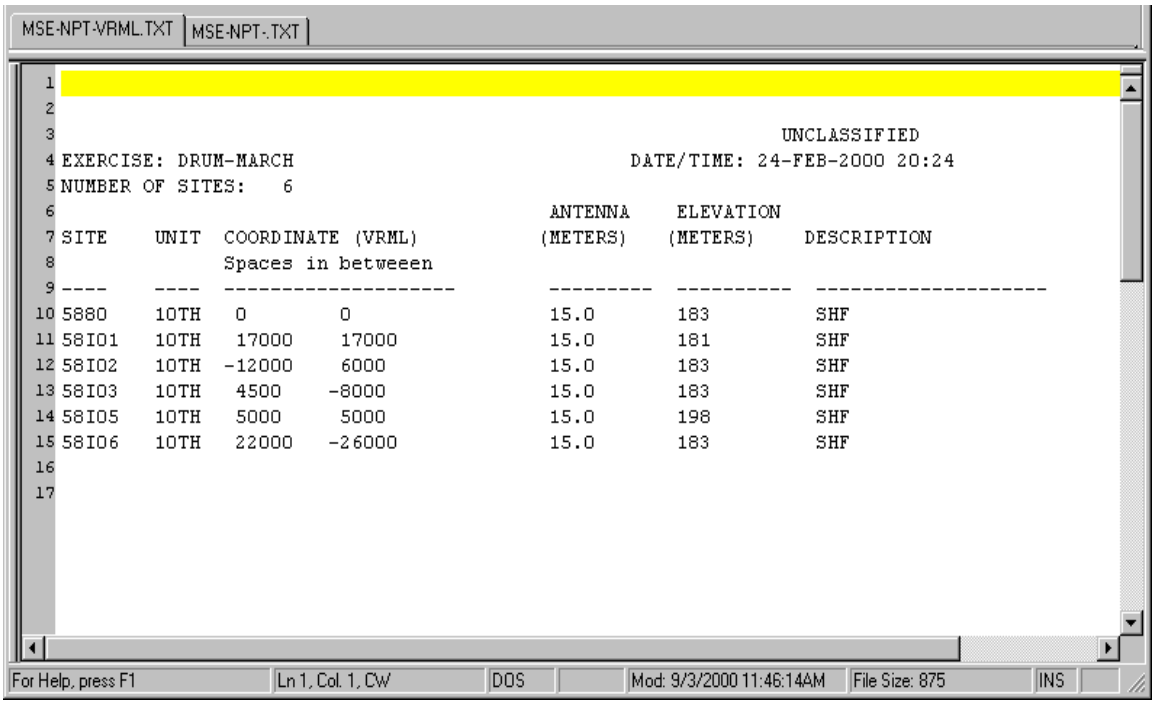


Figure 6.7: MSE-NPT File with VRML mission coordinates.

The MSE-NPT file reader then initiates the DIS-Java-VRML antenna start panel. The start panel allows the user to see the final configuration of the antenna visualization. It has options for the multicast address, multicast port, DIS site ID and DIS application. The user can select all of the antennas or just a portion of the file that has been loaded. There is also a button to start the visualization, a button to exit, and a button to rejoin the visualization already in progress. The rejoin button is useful when the user only wants to view a few antennas at first, and then later wants to add more antennas to the visualization. This panel is shown in Figure 6.8. Agent control functionality is not provided but remains a promising candidate for future work.

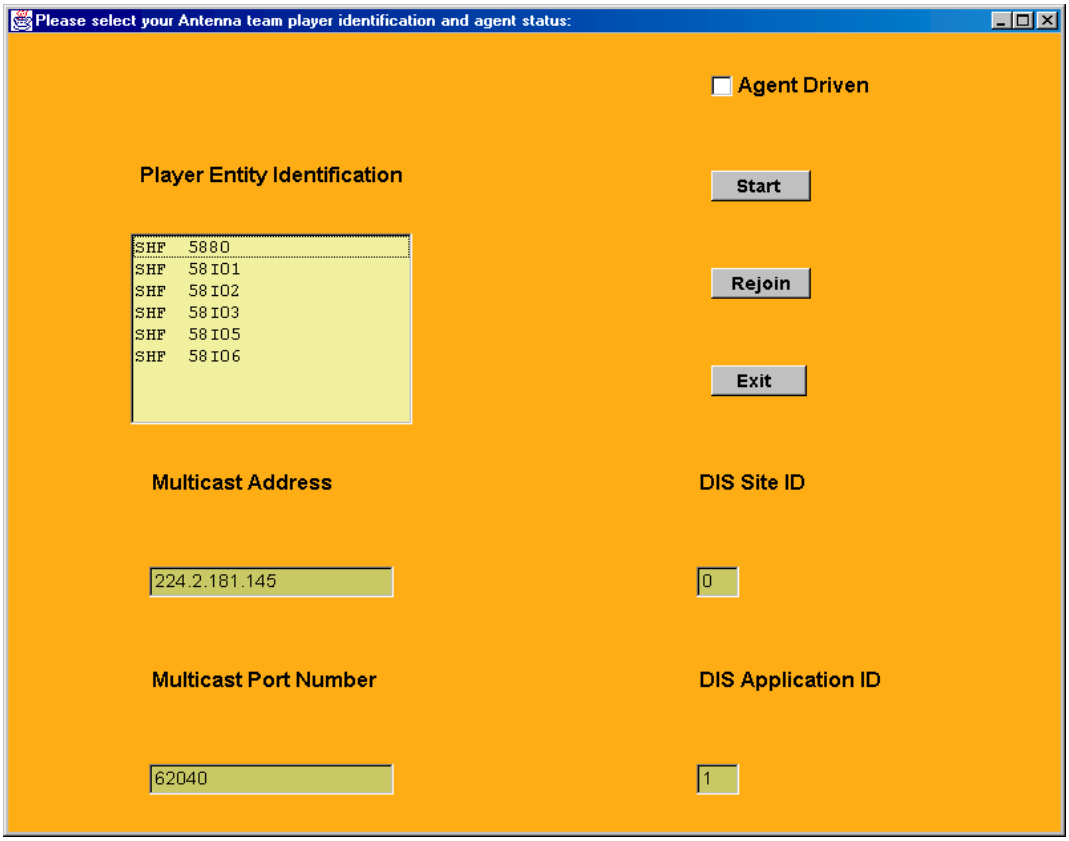


Figure 6.8: Antenna Start Panel with selection choices generated by file parser.

The connectivity between the VRML visualization and the network is enabled by the use of the antenna control panel. The control provides the user with means of interacting with the visualization. The panel is connected to both the entity state PDUs and radio communications family PDUs. This connectivity can be seen in Figures 6.9 and 6.10. The entity state PDUs affects the physical state of the antenna, including its location and orientation. The radio communications family PDUs updates the information visualization in the scene by sending or receiving *SignalPDUs*,

ReceiverPDUs or *TransmitterPDUs*. Sending PDUs corresponding to antenna panel setting is enabled by toggle boxes on the control panel.

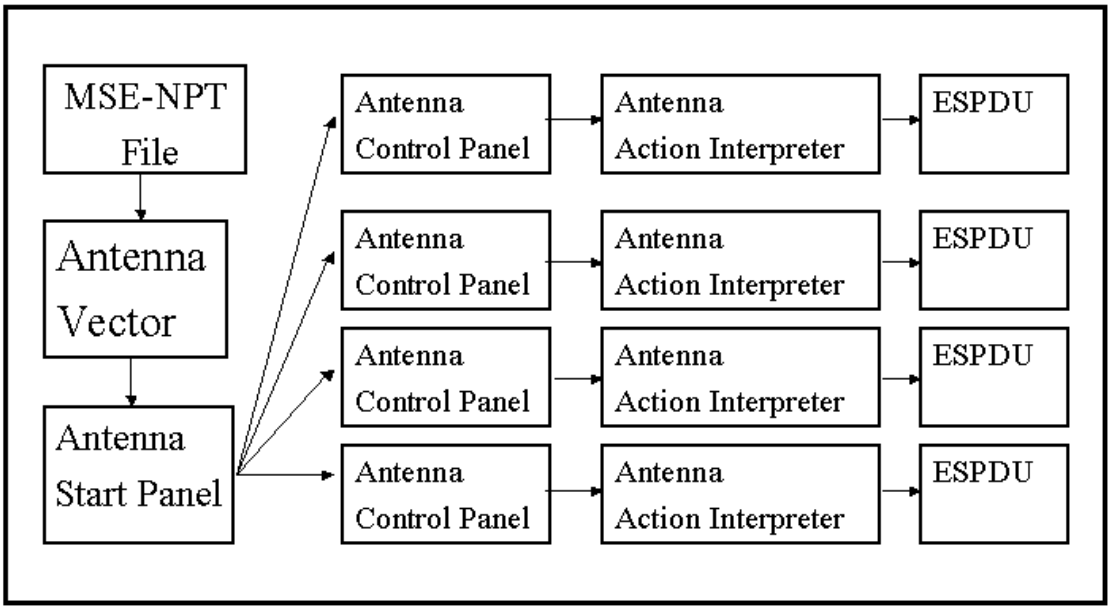


Figure 6.9: MSE-NPT file data flow for ESPDU state information.

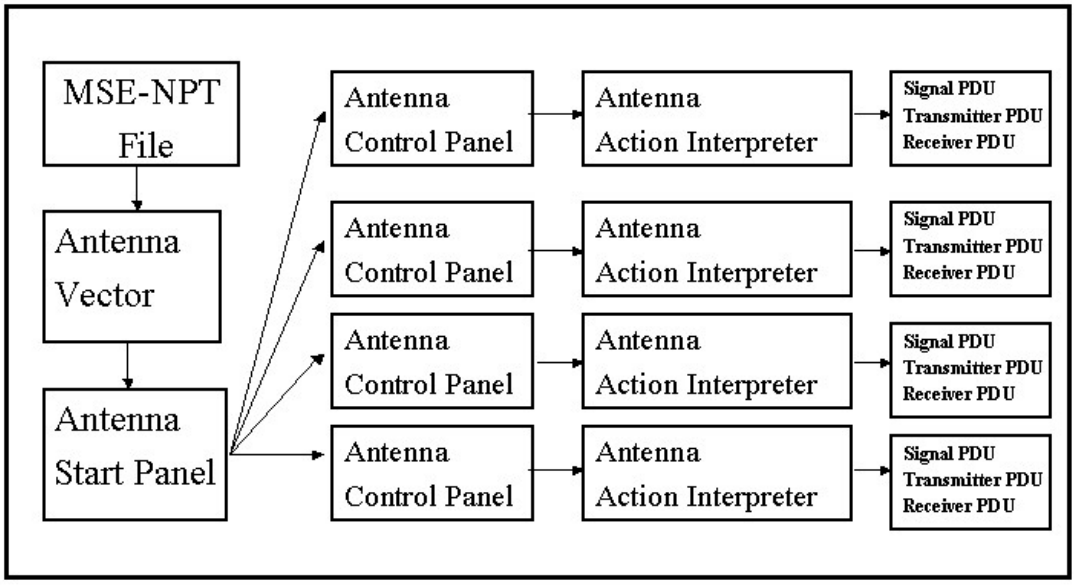


Figure 6.10: MSE-NPT File Data Flow for Radio Communications Family PDUs.

```
C:\vrtp\demo\helicopter>java demo.helicopter.StartPanel -pause 30
Pause before starting.............................
multicast timeToLive ttl=15
RTP headers prepended=false
entering actionPerformed() w/args...
   event = java.awt.event.ActionEvent[ACTION_PERFORMED,cmd=Antenna] on
button0
* * * Stand by start Antenna Pannel (listen 10 seconds for other
players...)
 TEST jButton_OpenFile_actionPerformed

Antenna 5880, 10TH, 0, 0, 15.0, 183, SHF    Good Line Read
Antenna 58I01, 10TH, 17000, 17000, 15.0, 181, SHF    Good Line Read
Antenna 58I02, 10TH, -12000, 6000, 15.0, 183, SHF    Good Line Read
Antenna 58I03, 10TH, 4500, -8000, 15.0, 183, SHF    Good Line Read
Antenna 58I05, 10TH, 5000, 5000, 15.0, 198, SHF    Good Line Read
Antenna 58I06, 10TH, 22000, -26000, 15.0, 183, SHF    Good Line Read
Got strategy of org.web3d.vrtp.security.SunSecurityStrategy
org.web3d.vrtp.net.DatagramStreamBuffer: *** constructor new instance
multicast org.web3d.net.DatagramStreamBuffer
    (224.2.181.145, 62040)
    (this should only appear once..)
Got strategy of org.web3d.vrtp.security.SunSecurityStrategy
org.web3d.vrtp.net.DatagramStreamBuffer: strategy.invokePrivilege(this,
"createMulticastSocket", [224.2.181.145/224.2.18
1.145 62040]);
org.web3d.vrtp.net.DatagramStreamBuffer: joined multicast address
224.2.181.145, port 62040
org.web3d.vrtp.net.DatagramStreamBuffer: setDEBUG false
BehaviorStreamBuffer: commencing datagramStreamBuffer.run ()...
org.web3d.vrtp.net.DatagramStreamBuffer: run:
strategy.invokePrivilege(this, "doRun");
org.web3d.vrtp.net.DatagramStreamBuffer: commence doRun ();
Listening for other vehicles........BehaviorStreamBuffer: shutdown ();
```

Figure 6.11: Antenna DIS-Java-VRML visualization startup sequence after the user has selected a startup MSE-NPT data file.

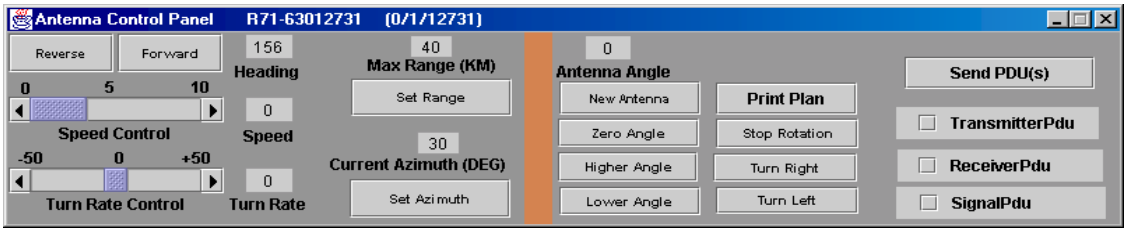


Figure 6.12: Antenna Control Panel allowing user to move, rotate, and send PDUs.

The control panel is the planners' interface to modify parameters for each antenna. They can manipulate the position and location of antennas, as well as send various DIS PDU types. Since this was a proof-of-concept implementation, some of the

functionality of the control panel is future work. When the user selects the check box by the PDU type, and then presses the send button, a PDU of that type is sent across the DIS Network. This PDU information is then visible to all of the participants in the visualization, as seen in Figure 6.13 and 6.14. These figures show a *SignalPDU's* state before and after transmission.

Each antenna also has a coverage dome associated with it. These domes can be changed by a field value that is sent into the VRML representation via a *Script* node. Figure 6.14 shows a green dome that has been changed by the VRML *eventIn* from the *Script* node to reflect the signal planning range of the antenna.

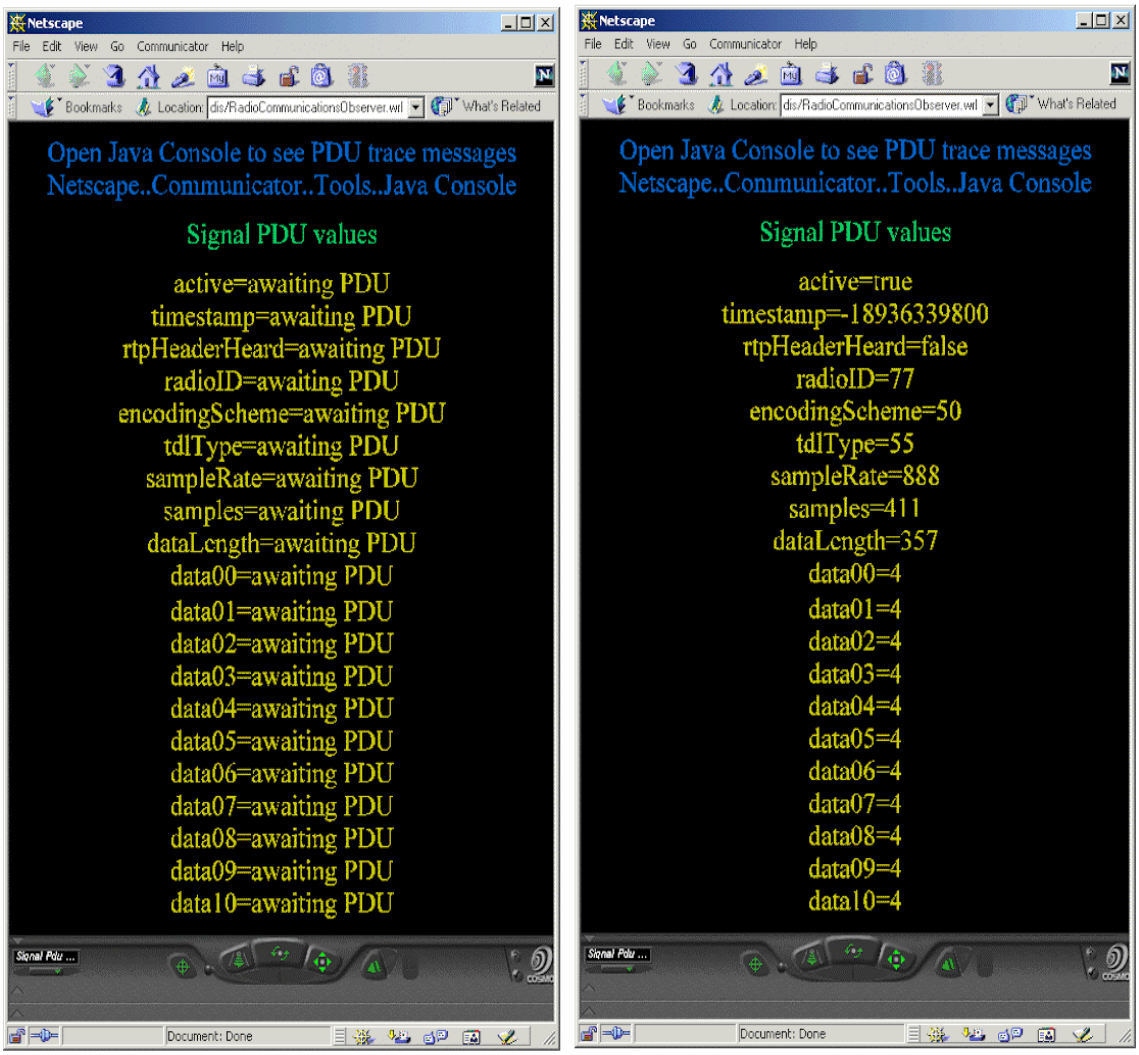


Figure 6.13: *SignalPDU* sent across the DIS Network Before and After Visualizations.

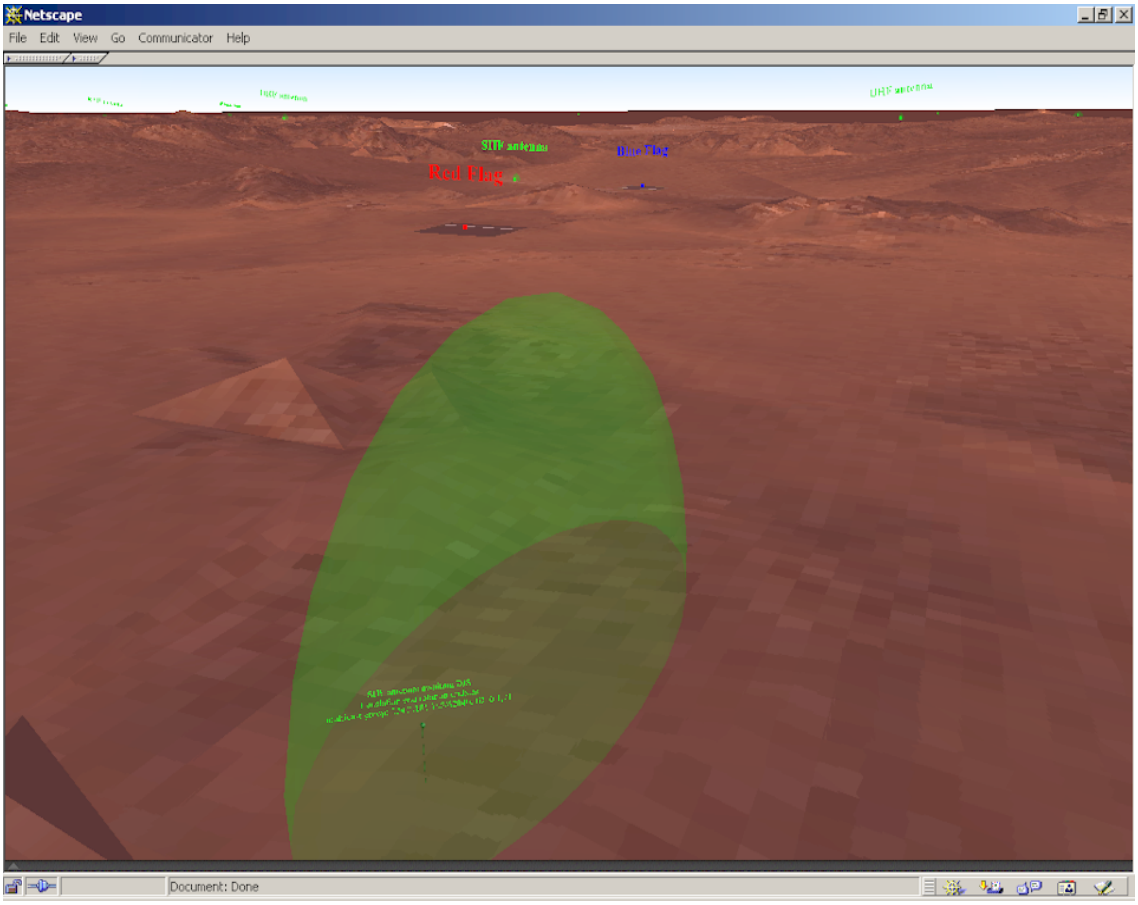


Figure 6.14: DIS-Java-VRML Antenna Signal Visualization far viewpoint.

C. VISUALIZATION RESULTS

The figures below are the result of the visualization, which can be generated with the DIS-Java-VRML visualization framework. The framework, in combination with many of the tools and technologies, generates realistic results.

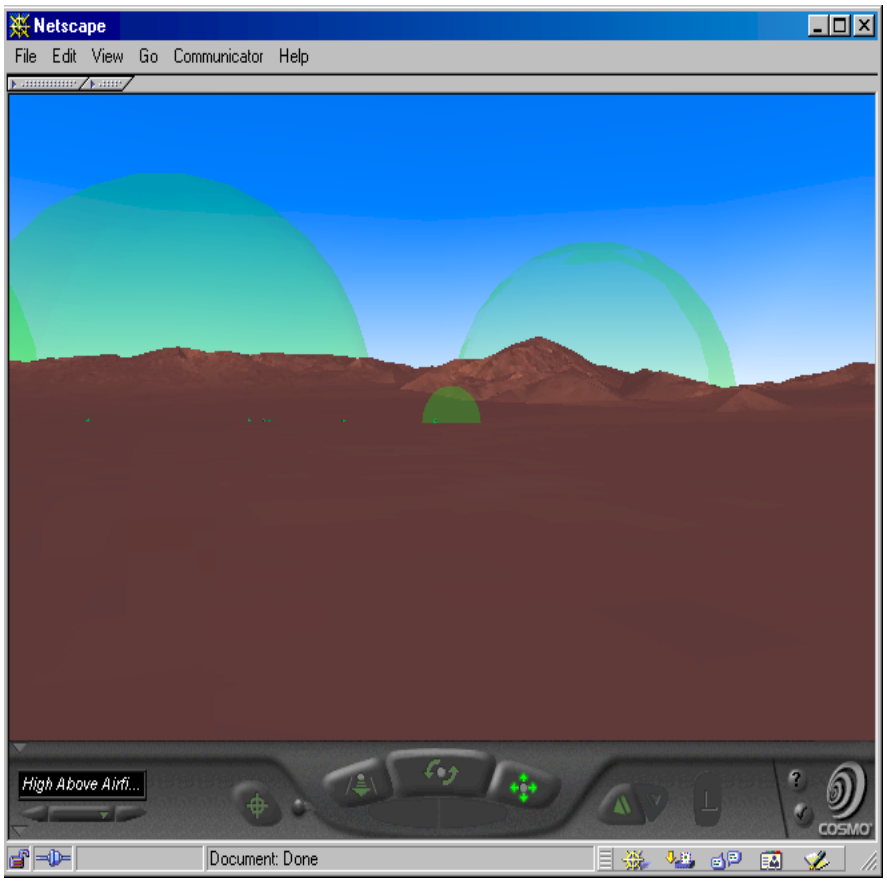


Figure 6.15: DIS-Java-VRML antenna signal visualization close in viewpoint showing 10 KM RAU antenna coverage.

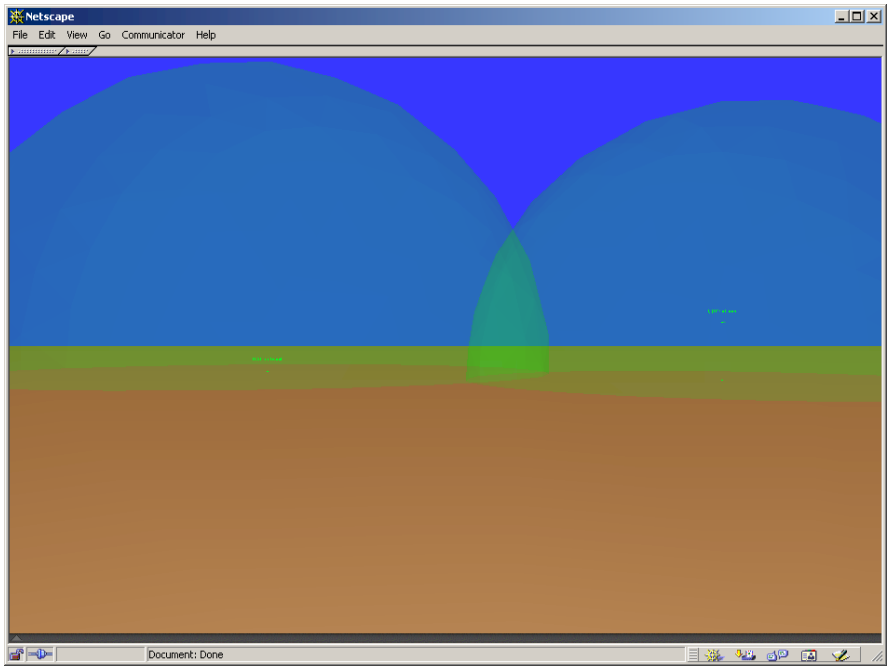


Figure 6.16: DIS-Java-VRML antenna signal visualization with 10 kilometer overlapping coverage domes.

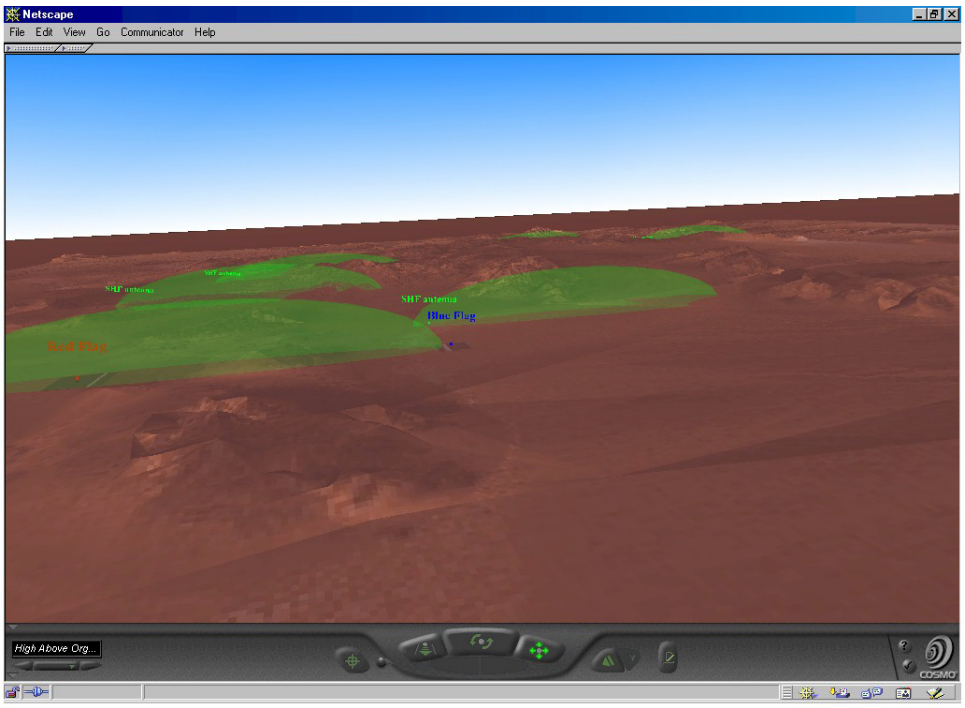


Figure 6.17: DIS-Java-VRML Antenna Signal Visualization with a viewpoint high above Fort Irwin California.

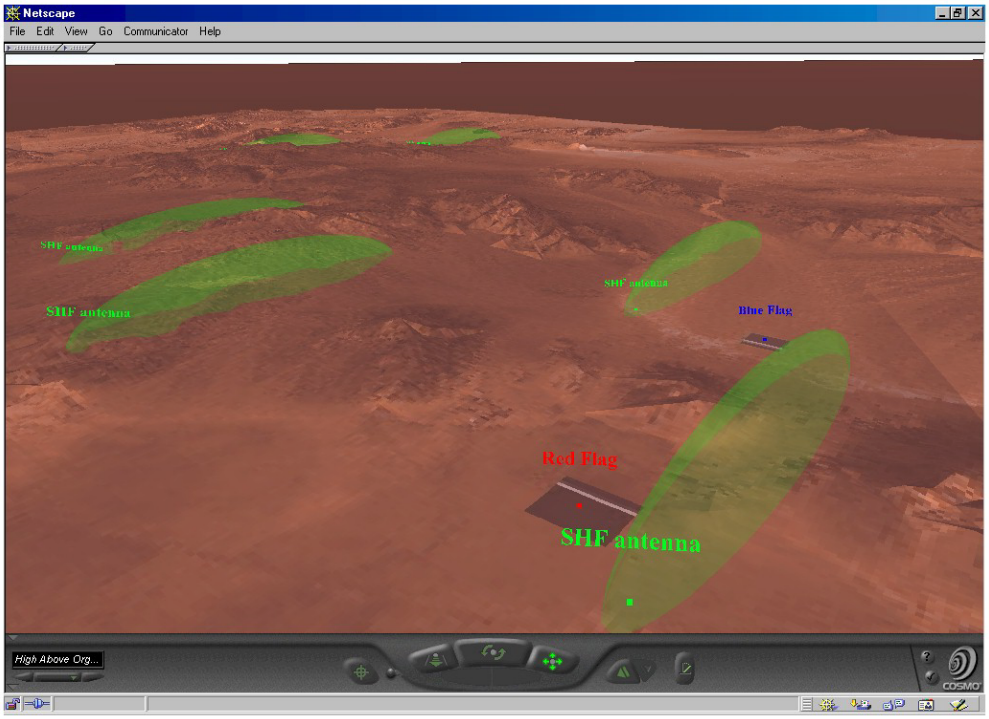


Figure 6.18: DIS-Java-VRML antenna signal visualization with a viewpoint high above the airfield.

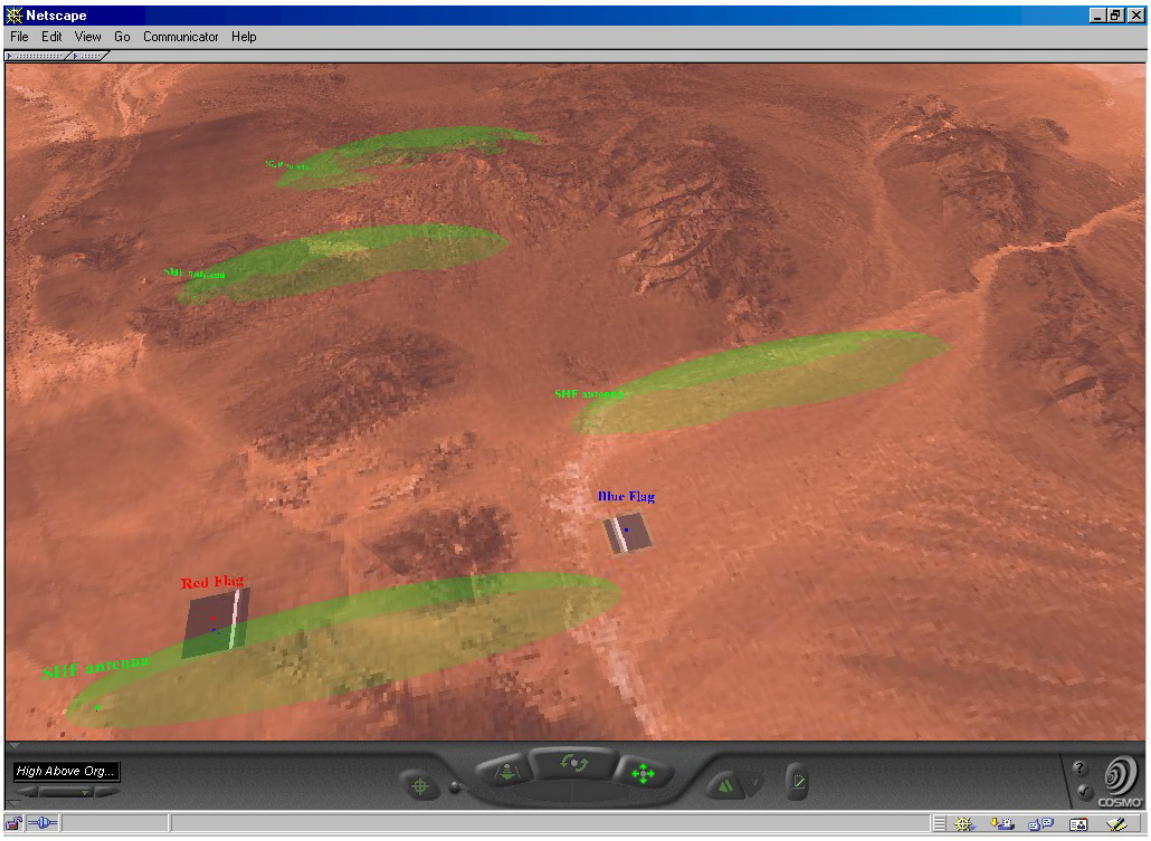


Figure 6.19: DIS-Java-VRML antenna signal visualization with a viewpoint 20 Km above the airfield.

D. PROPOSED USES OF RADIO COMMUNICATIONS VISUALIZATION

Representing a signal plan inside a virtual world can provide the planners with a number of advantages over current 2D methods. The virtual planning environment provides the entire battle space to the planner in 3D. By using this 3D space they can better gauge how the available equipment will best support a mission. By deploying the appropriate signals equipment (such as UHF, SHF, HF, or VHF radios) for the mission, the communicators have a better chance of providing reliable coverage, and better frequency de-confliction. By having a "big-picture view" of the battle space, the

communication planner can better inform the war fighter of the communications support plan in a geographical context.

E. SCIENTIFIC VISUALIZATION AND INTEROPERABILITY

For many years the Department of Defense (DoD) was a primary driver of computer-intensive technological developments. Over the past decade a shift from DoD to commercial development has occurred, and industry is now leading the technological revolution of the information age. Technologies are appearing daily which present the tools to solve the visualization and data interoperability problems found in many organizations. The X3D and VRML specifications are two recent technological developments that have major potential impact on scientific visualization, data structure, transmission, storage, and display. Additionally, scaling up becomes possible. "Virtual worlds can provide meaningful context to the mountains of content which currently exist in isolation without roads, links or order" (Brutzman, 1996).

F. SUMMARY

With current communications-planning requirements continuing to accelerate, something must to be done within the area of signals visualization to ease the information burden on the planner evaluating the operational requirements. Creative and innovative methodologies can provide the signal planner with an improved tool set, help them plan their mission more efficiently, and provide their commanders with a clear picture of the battle space. The signals visualization presented here demonstrates how visualization of MSE-NPT data can add to the perceptibility of the battle space.

VII. CONCLUSIONS AND RECOMMENDATIONS

A. INTRODUCTION

This chapter gathers the conclusions that have been reached relative to tactical communications visualization and visualization in general. A work of this nature leads to many opportunities for future work, which are described in the final section.

B. PRINCIPAL THESIS CONCLUSIONS

Construction of a 3D signals-visualization is feasible. It is possible to develop such a communications-visualization in a short period of time (in this case nine months), with software tools that are standards based and openly available. Using widely available and inexpensive 3D graphic rendering hardware and software makes it easier to reproduce signals visualizations. These visualizations allow for a greater number of planners to gain a deeper understanding of the MSE-NPT data, and also provides an avenue to visualize radio-wave propagation within a virtual battle space. The transformation of the MSE-NPT data into this visualization by using the DIS-Java-VRML software toolkit supports this conclusion.

C. SPECIFIC CONCLUSIONS

1. Tactical Communications Visualization

Tactical communications visualization is primarily performed with 2D tools, but these efforts are slowly transitioning to 3D tools. The DIS-Java-VRML toolkit is just

such a tool that can be used to facilitate this migration of signal planning data into 3D. The X3D-Edit authoring tool can rapidly generate offline 3D representations of military assemblages, which are easily integrated into the DIS-Java-VRML visualization. A DIS networked signal planning environment and virtual battle space enables planners to collaborate during the planning process. The integration of the DIS radio communications family PDUs into the DIS-Java-VRML DTD for X3D is a powerful tool for development of networked tactical environments.

2. Visualization

Visualization of numeric data in 3D can be a powerful medium for a decision maker to understand the overall picture of what the data really means and how it might impact them. By using visualization methods to see the interaction of the terrain data with the propagation data, a user can see significant interactions and instantly identify problems.

D. RECOMMENDATIONS FOR FURTHER WORK

1. Tactical Communications Visualization

The enhanced tactical communications visualization capabilities of the DIS-Java-VRML communications visualization framework suggests new development on an XML-based DTD for MSE-NPT Data files. This DTD, in conjunction with an XSL style sheet, allows the visualization to be transported across the network to a wider array of connected devices, such as wireless hand-held computers. Robust integration of

GeoVRML geographic referencing, and more accurate antenna position information, lead to greater flexibility. For example, data might be read not only from the MSE-NPT file but also from a large terrain server, thus allowing the use of additional resources in mission planning and rehearsal.

2. Model Validation

This visualization needs to be validated by units who are currently using the MSE-NPT for signal planning. This might be done by providing these types of visualizations to active units before they deploy to a major training facility (e.g. Fort Polk Louisiana or Fort Irwin California). Comparing statistical validation of the time to plan missions based on having the visualization (versus not having the visualization) may provide valuable insight in its usefulness. Also providing this visualization to non-signal planners might also yield additional feedback on the visualization.

3. Visualization

Another visualization possibility is to develop a 3D antenna placement system. Such a system takes the processed signal propagation data from the MSE-NPT system and through combined use of the signals visualization and knowledge of the planning environment, statistically calculates the optimum position for the antenna emitters.

The biggest, most important and most exciting task recommended for continued work is the exploration of new and more intuitive 3D representations for signal wave-propagation data. A detailed study of current signal-wave propagation data needs to be joined together with high-dimensional visual information concepts that utilize all the

capabilities of current 3D rendering hardware and software. A study of this type may prove to be of vital interest to the U.S. Army and military in general.

APPENDIX A. MSE-NPT File Reader Code

This is the code for the MSE-NPT file reader. It has three sections the main program, the Java Frame and the Java about box.

```
//Title:      Network Planning Terminal (NPT) File Reader
//Version:
//Copyright:  Copyright (c) 1999
//Author:     David W. Laflam
//Company:    NPS
//Description:NPT File Reader

package demo.helicopter;

import java.awt.*;
import java.awt.event.*;
import java.text.*;                             // for class DecimalFormat
import java.util.*;                             // for class Vector
import mil.navy.nps.dis.*;
import mil.navy.nps.testing.*;
import mil.navy.nps.util.*;
import mil.navy.nps.disEnumerations.*;
import javax.swing.UIManager;
import java.awt.*;
import java.io.* ;
import java.io.FileReader;

public class NetworkPlanningTerminalFileReader {
  boolean packFrame = false;

  //Construct the application
  public NetworkPlanningTerminalFileReader() {
    NetworkPlanningTerminalFileReaderFrame frame = new
    NetworkPlanningTerminalFileReaderFrame();

    //Validate frames that have preset sizes
    //Pack frames that have useful preferred size info, e.g. from their layout
    if (packFrame)
      frame.pack();
    else
      frame.validate();
    //Center the window
    Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
    Dimension frameSize = frame.getSize();
    if (frameSize.height > screenSize.height)
      frameSize.height = screenSize.height;
    if (frameSize.width > screenSize.width)
      frameSize.width = screenSize.width;
    frame.setLocation((screenSize.width - frameSize.width) / 2,
                      (screenSize.height - frameSize.height) / 2);
    frame.setVisible(true);
  }

  //Main method
  public static void main(String[] args) {
    try  {
   // for your system (Win or Mac)
   //   UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
   // Java metal look and feel
     UIManager.setLookAndFeel(UIManager.getCrossPlatformLookAndFeelClassName());
    }
    catch(Exception e) {
```

```
    }
    new NetworkPlanningTerminalFileReader();
  }
}
```

```
//Title:      Network Planning Terminal (NPT) File Reader
//Version:
//Copyright:  Copyright (c) 1999
//Author:     David W. Laflam
//Company:    NPS
//Description: NPT File Reader Frame, this is where the data files are parsed

package demo.helicopter;
import java.awt.*;
import java.awt.event.*;
import java.text.*;                     // for class DecimalFormat
import java.util.*;                     // for class Vector
import mil.navy.nps.dis.*;
import mil.navy.nps.testing.*;
import mil.navy.nps.util.*;
import mil.navy.nps.disEnumerations.*;
import javax.swing.*;
import javax.swing.UIManager;
import javax.swing.filechooser.*;
import java.io.* ;
import java.io.FileReader;
import java.io.FileWriter;
import java.util.StringTokenizer;


public class NetworkPlanningTerminalFileReaderFrame extends JFrame {
  JMenuBar menuBar1 = new JMenuBar();
  JMenu menuFile = new JMenu();
  JMenuItem menuFileExit = new JMenuItem();
  JMenu menuHelp = new JMenu();
  JMenuItem menuHelpAbout = new JMenuItem();
  JToolBar toolBar = new JToolBar();
  JButton jButton_OpenFile = new JButton();
  JButton jButton_WriteFile = new JButton();
  JButton jButton_Help = new JButton();
  JButton jButton_Exit = new JButton();

  ImageIcon image1;  // these are the gif files for the buttons
  ImageIcon image2;
  ImageIcon image3;
  ImageIcon image4;

  JLabel statusBar = new JLabel();
  JPanel jPanel1 = new JPanel();
  JLabel jLabel1 = new JLabel();


  private File file; // the file that I am reading

   ////////////// From the StartPanel.java //////////////////////
   public AntennaStartPanel asp; // the antenna start panel

   private static int exitHash = 0;
   private static int redHash = 0;
   private static int blueHash = 0;
   private static int antennaHash = 0;
   private static int timeToLive = 15; // default ttl convention (local campus)
   private static int sleepTime = 0;

   private static boolean rtpMatch = false;
   private static int lastEventCode;

   // prevent multiple copies of a live entity by listening first,
   //  > 5  seconds recommended
   // minimum listen time 2 sec needed or else panels will allow repeats...
   public  static final int multicastListenDelay = 8;
   public  static final int reportedMulticastListenDelay = multicastListenDelay + 2;
```

```
   ////////////// From the StartPanel.java ////////////////////

  //layout for the panel and frame
  GridBagLayout gridBagLayout1 = new GridBagLayout();
  GridBagLayout gridBagLayout2 = new GridBagLayout();

  //Construct the frame
  public NetworkPlanningTerminalFileReaderFrame() {
    enableEvents(AWTEvent.WINDOW_EVENT_MASK);
    try  {
      jbInit();
    }
    catch(Exception e) {
      e.printStackTrace();
    }
  }

  //Component initialization
  private void jbInit() throws Exception  {

    try {
       System.out.println ("initially trying /vrtp/demo/helicopter");
       image1 = new ImageIcon("/vrtp/demo/helicopter/readin.gif");
       image2 = new ImageIcon("/vrtp/demo/helicopter/process.gif");
       image3 = new ImageIcon("/vrtp/demo/helicopter/about.gif");
       image4 = new ImageIcon("/vrtp/demo/helicopter/exit.gif");
    }
    catch (Exception ex1)
    {
           // this worked on dave's system but not don's ???
           System.out.println ("now trying
demo.helicopter.NetworkPlanningTerminalFileReaderFrame.class.getResource(\"fileName.gif\")")
;
           try {
              image1 = new
ImageIcon(demo.helicopter.NetworkPlanningTerminalFileReaderFrame.class.getResource("readin.g
if"));
              image2 = new
ImageIcon(demo.helicopter.NetworkPlanningTerminalFileReaderFrame.class.getResource("process.
gif"));
              image3 = new
ImageIcon(demo.helicopter.NetworkPlanningTerminalFileReaderFrame.class.getResource("about.gi
f"));
              image4 = new
ImageIcon(demo.helicopter.NetworkPlanningTerminalFileReaderFrame.class.getResource("exit.gif
"));
           }
           catch (Exception ex2)
           {
              System.out.println ("error loading button files in
NetworkPlanningTerminalFileReaderFrame");
           }
    }

    this.getContentPane().setLayout(gridBagLayout1);
    this.setResizable(false);
    this.setSize(new Dimension(500, 115));
    this.setTitle("Network Planning Terminal File Reader");
    statusBar.setText(" ");
    menuFile.setText("File");
    menuFileExit.setText("Exit");
    menuFileExit.addActionListener(new ActionListener()  {

    public void actionPerformed(ActionEvent e) {
        fileExit_actionPerformed(e);
      }
    });
```

```
menuHelp.setText("Help");
menuHelpAbout.setText("About");

// for the about box use of inner class
menuHelpAbout.addActionListener(new ActionListener()  {
    public void actionPerformed(ActionEvent e) {
    helpAbout_actionPerformed(e);
  }
});
//=============================================================
jButton_OpenFile.setIcon(image1);
       jButton_OpenFile.addActionListener(new java.awt.event.ActionListener() {

  public void actionPerformed(ActionEvent e) {
    jButton_OpenFile_actionPerformed(e);
  }
});

jButton_OpenFile.addMouseListener(new java.awt.event.MouseAdapter() {

  public void mousePressed(MouseEvent e) {
    jButton_OpenFile_mousePressed(e);
  }
});

jButton_OpenFile.setToolTipText("Open File");
//=========================================================
jButton_WriteFile.setIcon(image2);
jButton_WriteFile.addActionListener(new java.awt.event.ActionListener() {

  public void actionPerformed(ActionEvent e) {
   // jButton_WriteFile_actionPerformed(e);
   ReadFile_actionPerformed(e);
  }
});

jButton_WriteFile.addMouseListener(new java.awt.event.MouseAdapter() {
  public void mousePressed(MouseEvent e) {
   // jButton_WriteFile_mousePressed(e);

  }
});


jButton_WriteFile.setToolTipText("Process & Close File");
//=============================================================
jButton_Help.setIcon(image3);

jButton_Help.addActionListener(new java.awt.event.ActionListener() {
     public void actionPerformed(ActionEvent e) {
    jButton_Help_actionPerformed(e);
  }
});

jButton_Help.setToolTipText("About");
//=====================================================================
jButton_Exit.setIcon(image4);
jButton_Exit.addActionListener(new java.awt.event.ActionListener() {

  public void actionPerformed(ActionEvent e) {
      System.exit(0);
  }
});

jButton_Exit.addMouseListener(new java.awt.event.MouseAdapter() {

  public void mousePressed(MouseEvent e) {
      System.exit(0);
```

```
      }
    });

    jButton_Exit.setToolTipText("Exit File");

//=================================================================   //

    toolBar.add(jButton_OpenFile);
    toolBar.add(jButton_WriteFile);
    toolBar.add(jButton_Help);
     toolBar.add(jButton_Exit);
    menuFile.add(menuFileExit);
    menuHelp.add(menuHelpAbout);
    menuBar1.add(menuFile);
    menuBar1.add(menuHelp);

    this.setJMenuBar(menuBar1);


this.getContentPane().add(toolBar, new GridBagConstraints(0, 0, 1, 1, 1.0,
0.0,GridBagConstraints.CENTER, GridBagConstraints.HORIZONTAL, new Insets(0, 0, 0, 0), 235,
0));

this.getContentPane().add(statusBar, new GridBagConstraints(0, 2, 1, 1, 0.0,
0.0,GridBagConstraints.WEST, GridBagConstraints.NONE, new Insets(0, 0, 0, 0), 389, 0));

/*
this.getContentPane().add(jPanel1, new GridBagConstraints(0, 1, 1, 1, 1.0,
1.0,GridBagConstraints.CENTER, GridBagConstraints.BOTH, new Insets(0, 0, 0, 0), 52, 74));

jPanel1.add(jLabel1, new GridBagConstraints(0, 1, 1, 1, 0.0, 0.0
,GridBagConstraints.WEST, GridBagConstraints.NONE, new Insets(39, 37, 74, 52), 17, 6));

jPanel1.add(ReadFile, new GridBagConstraints(0, 0, 1, 1, 0.0, 0.0,GridBagConstraints.CENTER,
GridBagConstraints.NONE, new Insets(55, 73, 0, 108), 56, 0));

*/
  }
  //File | Exit action performed
  public void fileExit_actionPerformed(ActionEvent e) {
    System.exit(0);
  }
  //=============================================================
  //Help | About action performed
  public void helpAbout_actionPerformed(ActionEvent e) {
    NetworkPlanningTerminalFileReaderFrameAboutBox dlg = new
NetworkPlanningTerminalFileReaderFrameAboutBox(this);
    Dimension dlgSize = dlg.getPreferredSize();
    Dimension frmSize = getSize();
    Point loc = getLocation();
    dlg.setLocation((frmSize.width - dlgSize.width) / 2 + loc.x, (frmSize.height -
dlgSize.height) / 2 + loc.y);
    dlg.setModal(true);
    dlg.show();
  }

  //=============================================================
  //Overridden so we can exit on System Close
  protected void processWindowEvent(WindowEvent e) {
    super.processWindowEvent(e);
    if(e.getID() == WindowEvent.WINDOW_CLOSING) {
      fileExit_actionPerformed(null);
    }
  }
  //=============================================================
  public Vector ReadFile_actionPerformed(ActionEvent e)
  {
    Vector antennaVector = new Vector ();
```

```
    int lineCounter = 0;


     try
     {
      FileReader fr = new FileReader(file);
       BufferedReader br = new BufferedReader(fr);
        String s;
       StringTokenizer thisLineTokens ;

         Antenna ant = null, ant1=null;
       while (( s = br.readLine()) != null)
       {


                 lineCounter++;
                 System.out.println("The line number is: " + lineCounter);

               if (lineCounter < 10)
               {
                      // do nothing
               }
               else
               {

                      thisLineTokens = new StringTokenizer(s);

                      while ( thisLineTokens.hasMoreTokens())
                      {
                                  String AntSystem1 = new
                      String(thisLineTokens.nextToken());
               System.out.println( AntSystem1);
                      String AntSystem2 = new String(thisLineTokens.nextToken());
               System.out.println( AntSystem2);
                      String AntSystem3 = new String(thisLineTokens.nextToken());
               System.out.println( AntSystem3);
                      String AntSystem4 = new String(thisLineTokens.nextToken());
               System.out.println( AntSystem4);
                      String AntSystem5 = new String(thisLineTokens.nextToken());
               System.out.println( AntSystem5);
                      String AntSystem6 = new String(thisLineTokens.nextToken());
                      System.out.println( AntSystem6);
                      String AntSystem7 = new String(thisLineTokens.nextToken());
                      System.out.println( AntSystem7);

                  if(ant != null)
                              {
                                   System.out.println("old ant=" + ant);
                                   ant1 =ant;
                              }
                              else
                              {
                                    System.out.println("ant is null");
                              }

ant = new Antenna(AntSystem1, AntSystem2, AntSystem3,
                  AntSystem4, AntSystem5, AntSystem6, AntSystem7);

System.out.println("new antenna = " + ant + " old ant is " + ant1);

antennaVector.add(ant);

)//end while

//System.out.println();
System.out.println(s);
}// end else
}//end while
```

```
// colse the file
fr.close();

}// end try



catch (Exception x)
{
System.err.println(" -- Bad Input, not a % --" + x );
}//end catch

    for (int ix = 0; ix < antennaVector.size(); ix++)
    {
    System.out.println(((Antenna)
       antennaVector.elementAt(ix)).toString());
    }

    // from start Panel
asp = new AntennaStartPanel (800, 650, antennaVector);
// for the Antenna

//    asp.addAntennas(antennaVector);   // add the antenna objects
this.setVisible(false);
asp.show();
asp.setTimeToLive (timeToLive);
//asp.setRtpHeaderEnabled (rtpMatch);
dispose();


return antennaVector;

} // end method ReadFile_actionPerformed()
//=========================================================
void jButton_WriteFile_actionPerformed(ActionEvent e) {

System.out.println(" TEST jButton_WriteFile_actionPerformed ");

try {

// need to put source .wrl VRML File Here
String source = " This will write out the file in the future \n"
                + " also will develop a MRGS to GeoVRML \n"
                + "conversion program. " ;


char buffer[] = new char[source.length()];
source.getChars(0, source.length(), buffer, 0);

FileWriter f1 = new FileWriter("file2.doc");
f1.write(buffer);
f1.close();

FileWriter f2 = new FileWriter("file3.txt");
f2.write(buffer);
f2.close();

}// end try
catch (Exception x) {
System.err.println(" -- Bad Input, not a % --");
}//end catch

System.out.println(" File Written Out to file2.doc and file3.txt ");

}// end  jButton_WriteFile_actionPerformed()

//=============================================================
```

```
void jButton_OpenFile_actionPerformed(ActionEvent e) {

System.out.println(" TEST jButton_OpenFile_actionPerformed ");
try {
          UIManager.setLookAndFeel(UIManager.getCrossPlatformLookAndFeelClassName());
        } // end try
    catch (Exception exc) {
          System.err.println("Error loading Java Look&Feel: " + exc);
        } // end catch

     JFileChooser fileChooser = new JFileChooser();
     fileChooser.setCurrentDirectory (new File
("/vrtp/demo/helicopter/NetworkPlanningTerminalData"));

      fileChooser.setFileSelectionMode(
         JFileChooser.FILES_ONLY );
      int result = fileChooser.showOpenDialog( this );

      // user clicked Cancel button on dialog
      if ( result == JFileChooser.CANCEL_OPTION )
         file = null;
      else
         file = fileChooser.getSelectedFile();

   } // end jButton_OpenFile_actionPerformed()

  //=============================================================
   void jButton_Help_actionPerformed(ActionEvent e) {
     //System.out.println(" TEST jButton_Help_actionPerformed ");
     NetworkPlanningTerminalFileReaderFrameAboutBox dlg = new
NetworkPlanningTerminalFileReaderFrameAboutBox(this);
     Dimension dlgSize = dlg.getPreferredSize();
     Dimension frmSize = getSize();
     Point loc = getLocation();
     dlg.setLocation((frmSize.width - dlgSize.width) / 2 + loc.x,
                     (frmSize.height - dlgSize.height) / 2 + loc.y);
     dlg.setModal(true);
     dlg.show();
   } // end jButton_Help_actionPerformed()

  //=============================================================
   void jButton_OpenFile_mousePressed(MouseEvent e) {
    // need to add code here to open the file menu

   }// end jButton_openFile_mousePressed ()
  //=====================================================================
   void jButton_WriteFile_mousePressed(MouseEvent e) {
    // need to add code here to write the file out

   } // end jButton_saveFile_mousePressed()
  //=============================================================
   void ReadFile_mousePressed(MouseEvent e) throws Exception {

   } // end ReadFile_mousePressed ()


}  // end class  NetworkPlanningTerminalFileReaderFrame()
```

```
//Title:      Network Planning Terminal (NPT) File Reader Application
//Version:
//Copyright:  Copyright (c) 1999
//Author:     David W. Laflam
//Company:    NPS
//Description:NPT File Reader About Box, gives a brief description of the Application

package demo.helicopter;

import java.awt.*;

import java.awt.event.*;
import java.text.*;                             // for class DecimalFormat
import java.util.*;                             // for class Vector
import mil.navy.nps.dis.*;
import mil.navy.nps.testing.*;
import mil.navy.nps.util.*;
import mil.navy.nps.disEnumerations.*;
import javax.swing.*;
import javax.swing.border.*;

public class NetworkPlanningTerminalFileReaderFrameAboutBox extends JDialog implements
ActionListener {

  JPanel panel1 = new JPanel();
  JPanel panel2 = new JPanel();
  JPanel insetsPanel1 = new JPanel();
  JPanel insetsPanel2 = new JPanel();
  JPanel insetsPanel3 = new JPanel();
  JButton button1 = new JButton();
  JLabel imageControl1 = new JLabel();
  ImageIcon imageIcon;
  JLabel label1 = new JLabel();
  JLabel label2 = new JLabel();
  JLabel label3 = new JLabel();
  JLabel label4 = new JLabel();
  JLabel label5 = new JLabel();
  JLabel label6 = new JLabel();
  BorderLayout borderLayout1 = new BorderLayout();
  BorderLayout borderLayout2 = new BorderLayout();
  FlowLayout flowLayout1 = new FlowLayout();
  FlowLayout flowLayout2 = new FlowLayout();
  GridLayout gridLayout1 = new GridLayout();
  String product = "Network Planning Terminal (NPT) File Reader ";
  String author = "Dave Laflam";
  String version = "Version 1.0";
  String revision = "20 September 2000";
  String copyright = "Naval Postgraduate School";
  String comments = "http://www.web3d.org/WorkingGroups/vrtp/dis-java-vrml";
  public NetworkPlanningTerminalFileReaderFrameAboutBox(Frame parent) {
    super(parent);
    enableEvents(AWTEvent.WINDOW_EVENT_MASK);
    try  {
      jbInit();
    }
    catch(Exception e) {
      e.printStackTrace();
    }
    //imageControl1.setIcon(imageIcon);
    pack();
  }

  private void jbInit() throws Exception  {
    //imageIcon = new ImageIcon(getClass().getResource("your image name goes here"));
    this.setTitle("About");
    setResizable(false);
    panel1.setLayout(borderLayout1);
```

```
    panel2.setLayout(borderLayout2);
    insetsPanel1.setLayout(flowLayout1);
    insetsPanel2.setLayout(flowLayout1);
    insetsPanel2.setBorder(new EmptyBorder(10, 10, 10, 10));
    gridLayout1.setRows(4);
    gridLayout1.setColumns(1);
    label1.setText(product);
    label2.setText(author);
    label3.setText(version);
    label4.setText(revision);
    label5.setText(copyright);
    label6.setText(comments);
    insetsPanel3.setLayout(gridLayout1);
    insetsPanel3.setBorder(new EmptyBorder(10, 60, 10, 10));
    button1.setText("OK");
    button1.addActionListener(this);
    panel1.setMinimumSize(new Dimension(500, 125));
    panel1.setPreferredSize(new Dimension(500, 125));
    insetsPanel2.add(imageControl1, null);
    panel2.add(insetsPanel2, BorderLayout.WEST);
    this.getContentPane().add(panel1, null);
    insetsPanel3.add(label1, null);
    insetsPanel3.add(label2, null);
    insetsPanel3.add(label3, null);
    insetsPanel3.add(label4, null);
    insetsPanel3.add(label5, null);
    insetsPanel3.add(label6, null);
    panel2.add(insetsPanel3, BorderLayout.CENTER);
    insetsPanel1.add(button1, null);
    panel1.add(insetsPanel1, BorderLayout.SOUTH);
    panel1.add(panel2, BorderLayout.NORTH);
  }

  protected void processWindowEvent(WindowEvent e) {
    if(e.getID() == WindowEvent.WINDOW_CLOSING) {
      cancel();
    }
    super.processWindowEvent(e);
  }

  void cancel() {
    dispose();
  }

  public void actionPerformed(ActionEvent e) {
    if(e.getSource() == button1) {
      cancel();
    }
  }
}
```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX B. DIS PDU CODE

Note: only the SignalPdu.java file is listed, the TransmitterPdu.java and ReceiverPdu.java file can be found in the DIS-Java-VRML distrobution.

```
/*
 File:          SignalPdu.java
 CVS Info:$Id: SignalPdu.java,v 1.0 2000/06/07 18:00:00 laflam Exp $
 Compiler:jdk 1.3
 */

package mil.navy.nps.dis;               // package for Naval Postgraduate School DIS Libaray
import mil.navy.nps.util.*;             // General-purpose utilities
import mil.navy.nps.disEnumerations.*;  // Enumerations for DIS
import java.lang.*;                     //
import java.util.*;                     // utility stuff we need
import java.io.*;                       // input/output for serialization


/**
 * Signal PDU for DIS.
 *
 *@version 1.0
 *@author <a href="mailto:dave@laflam.net">David W. Laflam</a> (<a
href="http://wwww.laflam.net/Dave">http://wwww.laflam.net/dave</a>)
 *<br>
 *@author <a href="mailto:brutzman@nps.navy.mil">Don Brutzman</a> (<a
href="http://web.nps.navy.mil/~brutzman">http://web.nps.navy.mil/~brutzman</a>)
 *
 *<dt><b>Location:</b>
 *<dd>Web:  <a href="http://www.web3d.org/WorkingGroups/vrtp/mil/navy/nps/dis/SignalPdu.java">
 *  http://www.web3d.org/WorkingGroups/vrtp/mil/navy/nps/dis/SignalPdu.java</a>
 *
 *<dd>or locally:  <a href="../../../../../../mil/navy/nps/dis/SignalPdu.java">
 *  ~/mil/navy/nps/dis/SignalPdu.java</a>
 *
 *<dt><b>Hierarchy Diagram:</b>
 *<dd><a href="../../../../../../dis-java-vrml/images/PduClassHierarchy.jpg"><IMG
SRC="../../../../../../dis-java-vrml/images/PduClassHierarchyButton.jpg" ALIGN=ABSCENTER
WIDTH=150 HEIGHT=21></a>
 *
 *<dt><b>Summary:</b>
 *<dd>The actual transmission of voice, audio or other data shall be communicated by issuing a
Signal PDU.
 *
 *<dt><b>Explanation:</b>
 *<dd>The Signal pdu denotes the reciving of a transmission from a radio.
 *  It inherits the header information from ProtocolDataUnit,
 *  an abstract class that contains assorted protocol information.
 *  It implements the IDs of what's transmitting a signalPDU.
 *  <P>
 *
 *  As with other PDUs, it knows how to serialize and deserialize itself
 *  from the wire. It also knows how to clone itself, and knows how to
 *  calculate its size when sent to the wire.<P>
 *
 *<dt><b>History:</b>
 *<dd>          15 May  2000
 *<dd>          17DAug00     /Dave Laflam  /Added toString method
 *<dd>          1Sep00       /Don Brutzman       /Added extra data elements
 *
 *<dt><b>References:</b>
 *<dd>          DIS Data Dictionary:
```

```
 *        <A
href="../../../../../../mil/navy/nps/disEnumerations/JdbeHtmlFiles/pdu/7c.htm">Signal PDU
(local)and
 *              <A href="http://SISO.sc.ist.ucf.edu/dis-dd/pdu/7c.htm">Signal PDU</a>
 *<dd>           DIS-Java-VRML Working Group: <a
href="http://www.web3d.org/WorkingGroups/vrtp/dis-java-vrml/">
 *        http://www.web3d.org/WorkingGroups/vrtp/dis-java-vrml/</a>
 *<dd>           DIS specification : IEEE 1278.1-1995, Section 5.3.8.2
 *
 *
 *@see ProtocolDataUnit
 *@see PduElement
 *@see SerializationInterface
 *@see RadioCommunicationsFamily
 *@see ReceiverPdu
 *@see TransmitterPdu
 *@see RadioCommunicationsPduScriptNode
 *
 */


public class SignalPdu extends RadioCommunicationsFamily
{


/**
  *
  *entity ID: This field shall identify the entity that is the source of the radio
transmission.
  *The source entity may either represent the radio itself or represent an entity (such as a
vehicle)
  *that contains the radio.
  *This field shall be represented by an Entity Identifier record (see 5.2.14).
  *
  * <dl>
  * <dt><b>Value:</b>
  * <dd>If the intended Entity ID is unknown, this field shall contain Entity ID_UNKNOWN.
  * <dt><b>Reference:</b>
  * <dd>  DIS Data Dictionary:
  *       <A href="../../../../../../mil/navy/nps/disEnumerations/JdbeHtmlFiles/pdu/2c.htm">
  *          Event Identifier Record (local)</A> and
  *       <a href="http://siso.sc.ist.ucf.edu/dis-dd/pdu/2c.htm">Event Identifier Record</a>
  * </dl>
  */

protected EntityID    entityID;     // (Site , Applications , Entity) are all 16 Bit Unsigned
Int
                                    // ID of entity that's doing the Transmission of the Signal

/**
  * radioID. This field shall identify a particular radio within a given entity.
  * This field shall be represented by a 16-bit unsigned integer. The Entity ID,
  * Radio ID pair associates each Signal PDU with the preceding Transmitter PDU
  * that contains the same Entity ID, Radio ID pair. The combination of Entity ID
  * and Radio ID uniquely identiÞes a particular radio within a simulation exercise.
  * Pg 115 (5.3.8.2)
  *<dl>
  *<dt><b>Reference:</b>
  *<dd>  DIS Data Dictionary:
  *<A href="../../../../../../mil/navy/nps/disEnumerations/JdbeHtmlFiles/pdu/79.htm">Event
Identifier Record (local)</A> and
  *<a href="http://siso.sc.ist.ucf.edu/dis-dd/pdu/79.htm">Event Identifier Record</a>
  *</dl>
  */
protected UnsignedShort          radioID;   //16-bit unsigned integer


/**
```

```
  * encodingScheme: This field shall specify the encoding used in the Data Þeld of this PDU.
  * The encoding scheme shall be composed of a 2-bit Þeld specifying the encoding class and
  * a 14-bit field specifying either the encoding type, or the number of TDL messages contained
  * in this Signal PDU(see table 57 pg 115).
  * <dl>
  * <dt><b>Reference:</b>
  * <dd>  DIS Data Dictionary:
  *       <A href="../../../../../../mil/navy/nps/disEnumerations/JdbeHtmlFiles/pdu/79.htm">
Event Identifier Record (local)</A> and
  *       <a href="http://siso.sc.ist.ucf.edu/dis-dd/pdu/79.htm">Event Identifier Record</a>
  * </dl>
  */

protected UnsignedShort          encodingScheme;    //16 Bit Enumeration


/**
  * tdlType:  This field shall specify the TDL Type as a 16-bit enumeration Þeld when the
encoding
  * class is the raw binary, audio, application-speciÞc, or database index representation of a
TDL
  * message. When the Data Þeld is not representing a TDL Message, this Þeld shall be set to
zero (see
  * Section 9 of EBV-DOC for enumeration of the TDL Type field).
  * <dl>
  * <dt><b>Reference:</b>
  * <dd>  DIS Data Dictionary:
  *       <A href="../../../../../../mil/navy/nps/disEnumerations/JdbeHtmlFiles/pdu/79.htm">
Event Identifier Record (local)</A> and
  *       <a href="http://siso.sc.ist.ucf.edu/dis-dd/pdu/79.htm">Event Identifier Record</a>
  * </dl>
  */


protected UnsignedShort tdlType ; // 16 bit enumeration

 /**
  * sampleRate:  This field shall specify either the sample rate in samples per second if the
encoding
  * class is encoded audio or, the data rate in bits per second for data transmissions. If the
encoding class
  * is database index, this Þeld shall be zero. This Þeld shall be represented by a 32-bit
unsigned integer.
  * <dl>
  * <dt><b>Reference:</b>
  * <dd>  DIS Data Dictionary:
  *       <A href="../../../../../../mil/navy/nps/disEnumerations/JdbeHtmlFiles/pdu/79.htm">
Event Identifier Record (local)</A> and
  *       <a href="http://siso.sc.ist.ucf.edu/dis-dd/pdu/79.htm">Event Identifier Record</a>
  * </dl>
  */
protected UnsignedInt  sampleRate;  // 32 bit integer     //look in UnsignedInt.java


 /**
  * dataLength:  This field shall specify the number of bits of digital voice audio or digital
data being
  * sent in this Signal PDU, and shall be represented by a 16-bit unsigned integer. If the
encoding class
  * is database index, the Data Length field shall contain the value 96.
  *
  * Currently hardwired to support 11 data elements.
  * <dl>
  * <dt><b>Reference:</b>
  * <dd>  DIS Data Dictionary:
  *       <A href="../../../../../../mil/navy/nps/disEnumerations/JdbeHtmlFiles/pdu/79.htm">
Event Identifier Record (local)</A> and
  *       <a href="http://siso.sc.ist.ucf.edu/dis-dd/pdu/79.htm">Event Identifier Record</a>
```

```
  * </dl>
  */
protected UnsignedShort dataLength;  // 16 bit integer    // look in UnsignedShort.java


/**
  * samples:  This field shall specify the number of samples in this PDU, and shall be
represented by a
  * 16-bit unsigned integer. If the encoding class is not encoded audio, this Þeld shall be
zero.
  * <dl>
  * <dt><b>Reference:</b>
  * <dd>  DIS Data Dictionary:
  *       <A href="../../../../../../mil/navy/nps/disEnumerations/JdbeHtmlFiles/pdu/79.htm">
Event Identifier Record (local)</A> and
  *       <a href="http://siso.sc.ist.ucf.edu/dis-dd/pdu/79.htm">Event Identifier Record</a>
  * </dl>
  */
protected UnsignedShort samples;  // 16 bit integer       // look in UnsignedShort.java


/**
  * data00:  This field shall specify the audio or digital data conveyed by the radio
transmission. The
  * interpretation of each Data field depends on the value of the encoding scheme [see 5.3.8.2
item d)]
  * and TDL Type [see 5.3.8.2 item e)] fields (page 116).
  * <dl>
  * <dt><b>Reference:</b>
  * <dd>  DIS Data Dictionary:
  *       <A href="../../../../../../mil/navy/nps/disEnumerations/JdbeHtmlFiles/pdu/79.htm">
Event Identifier Record (local)</A> and
  *       <a href="http://siso.sc.ist.ucf.edu/dis-dd/pdu/79.htm">Event Identifier Record</a>
  * </dl>
  */
protected UnsignedByte data00;  // 8 bit unsigned integer  // look in UnsignedByte.java

/**
  * data01:  This field shall specify the audio or digital data conveyed by the radio
transmission. The
  * interpretation of each Data field depends on the value of the encoding scheme [see 5.3.8.2
item d)]
  * and TDL Type [see 5.3.8.2 item e)] fields (page 116).
  * <dl>
  * <dt><b>Reference:</b>
  * <dd>  DIS Data Dictionary:
  *       <A href="../../../../../../mil/navy/nps/disEnumerations/JdbeHtmlFiles/pdu/79.htm">
Event Identifier Record (local)</A> and
  *       <a href="http://siso.sc.ist.ucf.edu/dis-dd/pdu/79.htm">Event Identifier Record</a>
  * </dl>
  */
protected UnsignedByte data01;  // 8 bit unsigned integer  // look in UnsignedByte.java

/**
  * data02:  This field shall specify the audio or digital data conveyed by the radio
transmission. The
  * interpretation of each Data field depends on the value of the encoding scheme [see 5.3.8.2
item d)]
  * and TDL Type [see 5.3.8.2 item e)] fields (page 116).
  * <dl>
  * <dt><b>Reference:</b>
  * <dd>  DIS Data Dictionary:
  *       <A href="../../../../../../mil/navy/nps/disEnumerations/JdbeHtmlFiles/pdu/79.htm">
Event Identifier Record (local)</A> and
  *       <a href="http://siso.sc.ist.ucf.edu/dis-dd/pdu/79.htm">Event Identifier Record</a>
  * </dl>
  */
protected UnsignedByte data02;  // 8 bit unsigned integer  // look in UnsignedByte.java
```

```
/**
  * data03:  This field shall specify the audio or digital data conveyed by the radio
transmission. The
  * interpretation of each Data field depends on the value of the encoding scheme [see 5.3.8.2
item d)]
  * and TDL Type [see 5.3.8.2 item e)] fields (page 116).
  * <dl>
  * <dt><b>Reference:</b>
  * <dd>  DIS Data Dictionary:
  *       <A href="../../../../../../mil/navy/nps/disEnumerations/JdbeHtmlFiles/pdu/79.htm">
Event Identifier Record (local)</A> and
  *       <a href="http://siso.sc.ist.ucf.edu/dis-dd/pdu/79.htm">Event Identifier Record</a>
  * </dl>
  */



protected UnsignedByte data03;  // 8 bit unsigned integer  // look in UnsignedByte.java

/**
  * data04:  This field shall specify the audio or digital data conveyed by the radio
transmission. The
  * interpretation of each Data field depends on the value of the encoding scheme [see 5.3.8.2
item d)]
  * and TDL Type [see 5.3.8.2 item e)] fields (page 116).
  * <dl>
  * <dt><b>Reference:</b>
  * <dd>  DIS Data Dictionary:
  *       <A href="../../../../../../mil/navy/nps/disEnumerations/JdbeHtmlFiles/pdu/79.htm">
Event Identifier Record (local)</A> and
  *       <a href="http://siso.sc.ist.ucf.edu/dis-dd/pdu/79.htm">Event Identifier Record</a>
  * </dl>
  */
protected UnsignedByte data04;  // 8 bit unsigned integer  // look in UnsignedByte.java

/**
  * data05:  This field shall specify the audio or digital data conveyed by the radio
transmission. The
  * interpretation of each Data field depends on the value of the encoding scheme [see 5.3.8.2
item d)]
  * and TDL Type [see 5.3.8.2 item e)] fields (page 116).
  * <dl>
  * <dt><b>Reference:</b>
  * <dd>  DIS Data Dictionary:
  *       <A href="../../../../../../mil/navy/nps/disEnumerations/JdbeHtmlFiles/pdu/79.htm">
Event Identifier Record (local)</A> and
  *       <a href="http://siso.sc.ist.ucf.edu/dis-dd/pdu/79.htm">Event Identifier Record</a>
  * </dl>
  */
protected UnsignedByte data05;  // 8 bit unsigned integer  // look in UnsignedByte.java

/**
  * data06:  This field shall specify the audio or digital data conveyed by the radio
transmission. The
  * interpretation of each Data field depends on the value of the encoding scheme [see 5.3.8.2
item d)]
  * and TDL Type [see 5.3.8.2 item e)] fields (page 116).
  * <dl>
  * <dt><b>Reference:</b>
  * <dd>  DIS Data Dictionary:
  *       <A href="../../../../../../mil/navy/nps/disEnumerations/JdbeHtmlFiles/pdu/79.htm">
Event Identifier Record (local)</A> and
  *       <a href="http://siso.sc.ist.ucf.edu/dis-dd/pdu/79.htm">Event Identifier Record</a>
  * </dl>
  */
protected UnsignedByte data06;  // 8 bit unsigned integer  // look in UnsignedByte.java
```

```
/**
  * data07:  This field shall specify the audio or digital data conveyed by the radio
transmission. The
  * interpretation of each Data field depends on the value of the encoding scheme [see 5.3.8.2
item d)]
  * and TDL Type [see 5.3.8.2 item e)] fields (page 116).
  * <dl>
  * <dt><b>Reference:</b>
  * <dd>  DIS Data Dictionary:
  *       <A href="../../../../../../mil/navy/nps/disEnumerations/JdbeHtmlFiles/pdu/79.htm">
Event Identifier
Record (local)</A> and
  *       <a href="http://siso.sc.ist.ucf.edu/dis-dd/pdu/79.htm">Event Identifier Record</a>
  * </dl>
  */
protected UnsignedByte data07;  // 8 bit unsigned integer  // look in UnsignedByte.java

/**
  * data08:  This field shall specify the audio or digital data conveyed by the radio
transmission. The
  * interpretation of each Data field depends on the value of the encoding scheme [see 5.3.8.2
item d)]
  * and TDL Type [see 5.3.8.2 item e)] fields (page 116).
  * <dl>
  * <dt><b>Reference:</b>
  * <dd>  DIS Data Dictionary:
  *       <A href="../../../../../../mil/navy/nps/disEnumerations/JdbeHtmlFiles/pdu/79.htm">
Event Identifier Record (local)</A> and
  *       <a href="http://siso.sc.ist.ucf.edu/dis-dd/pdu/79.htm">Event Identifier Record</a>
  * </dl>
  */
protected UnsignedByte data08;  // 8 bit unsigned integer  // look in UnsignedByte.java

/**
  * data09:  This field shall specify the audio or digital data conveyed by the radio
transmission. The
  * interpretation of each Data field depends on the value of the encoding scheme [see 5.3.8.2
item d)]
  * and TDL Type [see 5.3.8.2 item e)] fields (page 116).
  * <dl>
  * <dt><b>Reference:</b>
  * <dd>  DIS Data Dictionary:
  *       <A href="../../../../../../mil/navy/nps/disEnumerations/JdbeHtmlFiles/pdu/79.htm">
Event Identifier Record (local)</A> and
  *       <a href="http://siso.sc.ist.ucf.edu/dis-dd/pdu/79.htm">Event Identifier Record</a>
  * </dl>
  */
protected UnsignedByte data09;  // 8 bit unsigned integer  // look in UnsignedByte.java


/**
  * data10:  This field shall specify the audio or digital data conveyed by the radio
transmission. The
  * interpretation of each Data field depends on the value of the encoding scheme [see 5.3.8.2
item d)]
  * and TDL Type [see 5.3.8.2 item e)] fields (page 116).
  * <dl>
  * <dt><b>Reference:</b>
  * <dd>  DIS Data Dictionary:
  *       <A href="../../../../../../mil/navy/nps/disEnumerations/JdbeHtmlFiles/pdu/79.htm">
Event Identifier Record (local)</A> and
  *       <a href="http://siso.sc.ist.ucf.edu/dis-dd/pdu/79.htm">Event Identifier Record</a>
  * </dl>
  */
protected UnsignedByte data10;  // 8 bit unsigned integer  // look in UnsignedByte.java

/**
```

```
  *Constant value--size of Fire PDU with header. Here:
  *<code>sizeOf = 256 bytes</code>
  *
  * Total Signal Size = 256 + Data Length + 0 to 31, padding bits to increase the total
  * Signal Size to a multiple of 32 Bits.
  * Current Size 264 bits
  */

public final static int     sizeOf = 264 + 9*8;       // is this the PDU Size Total or is this
number in the header
                                                    // size of object as written to wire

/**
 *Default constructor
 * - creates  eventID, radioID, encodingScheme,
 *   tdlType, sampleRate, dataLength, samples, data00
 *
 * - fills with zeros for all values of the following parameters:
 *
 */

public SignalPdu()
{
   super.setPduType(PDUTypeField.SIGNAL);         // inherited from the super class
   entityID  = new EntityID();                          // 3 field (site,app,enity) 16-bit
unsigned integer
   radioID = new UnsignedShort(0);                      // 16-bit unsigned integer
   encodingScheme = new UnsignedShort(0);         // 16-bit enumeration
   tdlType = new UnsignedShort(0);                      // 16-bit enumeration
   sampleRate = new UnsignedInt(0);                     // 32-bit integer
   dataLength = new UnsignedShort(11);            // 16-bit integer
   samples = new UnsignedShort(0);                      // 16-bit integer
   data00 = new UnsignedByte(0);                        // 8-bit unsigned integer
   data01 = new UnsignedByte(0);                        // 8-bit unsigned integer
   data02 = new UnsignedByte(0);                        // 8-bit unsigned integer
   data03 = new UnsignedByte(0);                        // 8-bit unsigned integer
   data04 = new UnsignedByte(0);                        // 8-bit unsigned integer
   data05 = new UnsignedByte(0);                        // 8-bit unsigned integer
   data06 = new UnsignedByte(0);                        // 8-bit unsigned integer
   data07 = new UnsignedByte(0);                        // 8-bit unsigned integer
   data08 = new UnsignedByte(0);                        // 8-bit unsigned integer
   data09 = new UnsignedByte(0);                        // 8-bit unsigned integer
   data10 = new UnsignedByte(0);                        // 8-bit unsigned integer

   return;
} // end public SignalPdu()


/**
 * Make a copy of the object. This requires a deep copy, so we don't have two
 * objects sharing pointers to the same data.
 * @return a new Signal PDU entity
 */


public Object clone()
{

 SignalPdu    newSignalPdu = (SignalPdu)super.clone(); // this will inherit from the super
class

 newSignalPdu.setEntityID(this.getEntityID());
 newSignalPdu.setRadioID(this.getRadioID());
 newSignalPdu.setEncodingScheme(this.getEncodingScheme());
 newSignalPdu.setTDLType(this.getTDLType());
 newSignalPdu.setSampleRate(this.getSampleRate());
 newSignalPdu.setDataLength(this.getDataLength());
 newSignalPdu.setSamples(this.getSamples());
```

```
 newSignalPdu.setData00(this.getData00());
 newSignalPdu.setData01(this.getData01());
 newSignalPdu.setData02(this.getData02());
 newSignalPdu.setData03(this.getData03());
 newSignalPdu.setData04(this.getData04());
 newSignalPdu.setData05(this.getData05());
 newSignalPdu.setData06(this.getData06());
 newSignalPdu.setData07(this.getData07());
 newSignalPdu.setData08(this.getData08());
 newSignalPdu.setData09(this.getData09());
 newSignalPdu.setData10(this.getData10());

 return newSignalPdu;
} // end public Object clone()



/**
 * Serialize and write out the output stream, order is important here since
 * it needs to conform to the DIS standard
 * @exception RuntimeException when IO error occurs.
 */



public void serialize(DataOutputStream outputStream)
{
    super.serialize(outputStream);      // write out header info

//Note: you do not need a try and catch in this method, these are in the entityId.java
//which has it in site.java go back up the tree

 //   try
 //   {
       entityID.serialize(outputStream);
       radioID.serialize(outputStream);
       encodingScheme.serialize(outputStream);
       tdlType.serialize(outputStream);
       sampleRate.serialize(outputStream);
   dataLength.serialize(outputStream);
   samples.serialize(outputStream);
   data00.serialize(outputStream);
   data01.serialize(outputStream);
   data02.serialize(outputStream);
   data03.serialize(outputStream);
   data04.serialize(outputStream);
   data05.serialize(outputStream);
   data06.serialize(outputStream);
   data07.serialize(outputStream);
   data08.serialize(outputStream);
   data09.serialize(outputStream);
   data10.serialize(outputStream);

       // padding.serialize(outputStream);
       // outputStream.writeFloat(receiverPower);       // since this is a primitive value

//    }
//    catch (IOException ioError)
//    {
//        throw new
//            RuntimeException("Exception in SignalPdu.serialize, error writing to wire.");
//    }
    return;
}// end public void serialize()
```

```
/**
 * Deserialize the input stream, and order is important here, since we need to
 * read in the same order as specified by the DIS standard
 * @exception RuntimeException when IO error occurs.
 */

public void deSerialize(DataInputStream inputStream)
{
    super.deSerialize(inputStream);     // read in all the header info

//    try
//    {
        entityID.deSerialize(inputStream);
        radioID.deSerialize(inputStream);
        encodingScheme.deSerialize(inputStream);
        tdlType.deSerialize(inputStream);
        sampleRate.deSerialize(inputStream);
   dataLength.deSerialize(inputStream);
   samples.deSerialize(inputStream);
   data00.deSerialize(inputStream);
   data01.deSerialize(inputStream);
   data02.deSerialize(inputStream);
   data03.deSerialize(inputStream);
   data04.deSerialize(inputStream);
   data05.deSerialize(inputStream);
   data06.deSerialize(inputStream);
   data07.deSerialize(inputStream);
   data08.deSerialize(inputStream);
   data09.deSerialize(inputStream);
   data10.deSerialize(inputStream);

     //   receiverPower = inputStream.readFloat();   // since this is a primitive value
     //   tranmitterEntityID.deSerialize(inputStream);

//    }
//    catch (IOException ioError)
//    {
//        throw new
//            RuntimeException("Exception in SignalPdu.deSerialize, error reading from wire.");
//    }
}// end public void deSerialize()


/**
 * Returns the length of the entity
 * @return an integer length of the entity
 */

public int length()
{
    return sizeOf;          // EntityTypes are this long, always.  This is the 288
}// end public int length()


/**
 * Returns the PDU name - Signal PDU
 * @return a string "Signal PDU"
 */


public String pduName()
{
  return new String("Signal PDU");
} // end public String pduName()
```

```
/**
 * Print the values of the following object out, with correct level of
 * indentation on the page.
 * EntityID, RadioID, Encoding Scheme, TDL Type, Sample Rate, Data Length, Samples, data00
 *
 */

public void printValues(int indentLevel, PrintStream printStream)
{

  StringBuffer indent = ProtocolDataUnit.getPaddingOfLength(indentLevel);
  int          idx, superclassIndent = indentLevel;

    printStream.println();
    printStream.println("Signal PDU-");

    // ugly wart: get the superclass over to the left a couple pixels, if we have any to spare,
    // so the header info will be indented a bit less.

    if(superclassIndent > 0)
      superclassIndent -= 1;

     super.printValues(superclassIndent, printStream);
     entityID.printValues(indentLevel, printStream);
     printStream.println(indent + "radioID: " +  radioID);  // print the primitive type
     printStream.println(indent + "encodingScheme: " +  encodingScheme);  // print the
primitive type
     printStream.println(indent + "tdlType: " + tdlType);
     printStream.println(indent + "sampleRate: " + sampleRate);
     printStream.println(indent + "dataLength: " + dataLength);
     printStream.println(indent + "samples: " + samples);
     printStream.println(indent + "data00: " + data00);

   // tranmitterEntityID.printValues(indentLevel, printStream);
   // printStream.println(indent + "transmitterRadioID: " + transmitterRadioID);  // print the
primitive type

    return;
} // end public void printValues()

//Accessor methods ( the Set and Get Methods)

/**
 * Gets entity ID.
 * Each Entity in a given exercise executing on a DIS application shall be assigned an Entity
Identifier Record
 * Unique to the exercise.
 * @return a clone of the firing entity ID
 */

public EntityID getEntityID()
{
   return (EntityID)entityID.clone();
}

/**
 * Sets entity ID
 * Each Entity in a given exercise executing on a DIS application shall be assigned an Entity
Identifier Record
 * Unique to the exercise.
 * @param pFiringEntityID the firing entity ID
 */

public void setEntityID(EntityID pEntityID)
{
   entityID = pEntityID;
}
```

```
/**
 *Sets setEntityID(short pSiteID, short pApplicationID, short pEntityID),accessor method.
 *will create an new EntityID = entityID
 *This field shall identify the entity issuing the PDU,
 * and shall be represented by the PDU Header Record (see 5.2.24)
 */
public void setEntityID(short pSiteID, short pApplicationID, short pEntityID)
{ entityID = new EntityID(pSiteID, pApplicationID, pEntityID);
}


/**
 * Gets getRadio ID.
 * Each Entity in a given exercise executing on a DIS application shall be assigned an Entity
Identifier Record
 * Unique to the exercise.
 * @return a clone of the target entity ID
 */

public UnsignedShort getRadioID()
{
   return (UnsignedShort)radioID.clone();
}

/**
 * Sets setRadio ID.
 * Each Entity in a given exercise executing on a DIS application shall be assigned an Entity
Identifier Record
 * Unique to the exercise.
 * @param pRadioID target entity ID value
 */



public void setRadioID(UnsignedShort pRadioID)
{
   radioID = pRadioID;
}

/**
 * Gets the EncodingScheme.
 * Each Entity in a given exercise executing on a DIS application shall be assigned an Entity
Identifier Record
 * Unique to the exercise.
 * @return a clone of a EncodingScheme
 */

public UnsignedShort getEncodingScheme()
{
          return (UnsignedShort)encodingScheme.clone();
}

/**
 * Sets the EncodingScheme
 * @param pEncodingScheme a EncodingScheme
 */

public void setEncodingScheme(UnsignedShort pEncodingScheme)
{
   encodingScheme = pEncodingScheme;
}


// no need for a get and set for the padding //DWL
```

```
/**
 * Sets TDLType.
 * Each Entity in a given exercise executing on a DIS application shall be assigned an Entity
Identifier Record
 * Unique to the exercise.
 * @param pTDLType target entity ID value
 */
public void setTDLType (UnsignedShort pTDLType)
{
   tdlType = pTDLType;
}


/**
 * Gets the TDLType.
 * Each Entity in a given exercise executing on a DIS application shall be assigned an Entity
Identifier Record
 * Unique to the exercise.
 * @return a clone of a TDLType
 */
public UnsignedShort getTDLType()
{
    return (UnsignedShort)tdlType.clone();
}


/**
 * Sets SampleRate
 * Each Entity in a given exercise executing on a DIS application shall be assigned an Entity
Identifier Record
 * Unique to the exercise.
 * @param pSampleRate target entity ID value
 */
public void setSampleRate(UnsignedInt pSampleRate)
{
          sampleRate = pSampleRate;
}


/**
 * Gets the SampleRate.
 * Each Entity in a given exercise executing on a DIS application shall be assigned an Entity
Identifier Record
 * Unique to the exercise.
 * @return a clone of a SampleRate
 */
public UnsignedInt getSampleRate()
{
   return (UnsignedInt)sampleRate.clone();
}


/**
 * Sets DataLength
 * Each Entity in a given exercise executing on a DIS application shall be assigned an Entity
Identifier Record
 * Unique to the exercise.
 * @param pDataLength target entity ID value
 */
public void setDataLength(UnsignedShort pDataLength)
{
   dataLength = pDataLength;
}
```

```
/**
 * Gets the DataLength.
 * Each Entity in a given exercise executing on a DIS application shall be assigned an Entity
Identifier Record
 * Unique to the exercise.
 * @return a clone of a DataLength
 */
public UnsignedShort getDataLength()
{
   return (UnsignedShort)dataLength.clone();
}


/**
 * Sets Samples
 * Each Entity in a given exercise executing on a DIS application shall be assigned an Entity
Identifier Record
 * Unique to the exercise.
 * @param pSamples target entity ID value
 */
public void setSamples(UnsignedShort pSamples)
{
   samples = pSamples;
}


/**
 * Gets the Samples.
 * Each Entity in a given exercise executing on a DIS application shall be assigned an Entity
Identifier Record
 * Unique to the exercise.
 * @return a clone of a Samples
 */
public UnsignedShort getSamples()
{
   return (UnsignedShort)samples.clone();
}


/**
 * accessor method
 * @param raw 8-bit data
 */
public void setData00(UnsignedByte pdata00)
{
   data00 = pdata00;
}


/**
 * accessor method
 * @param raw 8-bit data
 */
public void setData01(UnsignedByte pdata01)
{
   data01 = pdata01;
}
```

```
/**
 * accessor method
 * @param raw 8-bit data
 */
public void setData02(UnsignedByte pdata02)
{
   data02 = pdata02;
}


/**
 * accessor method
 * @param raw 8-bit data
 */
public void setData03(UnsignedByte pdata03)
{
   data03 = pdata03;
}


/**
 * accessor method
 * @param raw 8-bit data
 */
public void setData04(UnsignedByte pdata04)
{
   data04 = pdata04;
}


/**
 * accessor method
 * @param raw 8-bit data
 */
public void setData05(UnsignedByte pdata05)
{
   data05 = pdata05;
}


/**
 * accessor method
 * @param raw 8-bit data
 */
public void setData06(UnsignedByte pdata06)
{
   data06 = pdata06;
}



/**
 * accessor method
 * @param raw 8-bit data
 */
public void setData07(UnsignedByte pdata07)
{
   data07 = pdata07;
}


/**
 * accessor method
 * @param raw 8-bit data
 */
public void setData08(UnsignedByte pdata08)
```

```
{
   data08 = pdata08;
}


/**
 * accessor method
 * @param raw 8-bit data
 */
public void setData09(UnsignedByte pdata09)
{
   data09 = pdata09;
}


/**
 * accessor method
 * @param raw 8-bit data
 */
public void setData10(UnsignedByte pdata10)
{
   data10 = pdata10;
}


/**
 * accessor method
 * @return a clone of a data00
 */
public UnsignedByte getData00()
{
   return (UnsignedByte)data00.clone();
}

/**
 * accessor method
 * @return a clone of a data01
 */
public UnsignedByte getData01()
{
   return (UnsignedByte)data01.clone();
}

/**
 * accessor method
 * @return a clone of a data02
 */
public UnsignedByte getData02()
{
   return (UnsignedByte)data02.clone();
}

/**
 * accessor method
 * @return a clone of a data03
 */
public UnsignedByte getData03()
{
   return (UnsignedByte)data03.clone();
}

/**
 * accessor method
 * @return a clone of a data04
 */
public UnsignedByte getData04()
{
   return (UnsignedByte)data04.clone();
```

```
}

/**
 * accessor method
 * @return a clone of a data05
 */
public UnsignedByte getData05()
{
   return (UnsignedByte)data05.clone();
}

/**
 * accessor method
 * @return a clone of a data06
 */
public UnsignedByte getData06()
{
   return (UnsignedByte)data06.clone();
}



/**
 * accessor method
 * @return a clone of a data07
 */
public UnsignedByte getData07()
{
   return (UnsignedByte)data07.clone();
}

/**
 * accessor method
 * @return a clone of a data08
 */
public UnsignedByte getData08()
{
   return (UnsignedByte)data08.clone();
}

/**
 * accessor method
 * @return a clone of a data09
 */
public UnsignedByte getData09()
{
   return (UnsignedByte)data09.clone();
}

/**
 * accessor method
 * @return a clone of a data10
 */
public UnsignedByte getData10()
{
   return (UnsignedByte)data10.clone();
}

/**
  * String toString
  * Used for debuging
  * System.out.println("Signal Object.  = " + signal);
  * This print out all values for the fields for the NEW Signal object
  */

 public String toString ()
 {
```

```
   String result;
   result = "\nEntityID = " + entityID + " \nRadioID = " + radioID
            + "\nEncodingScheme = " + encodingScheme
            + "\nTdlType = " + tdlType + "\nSampleRate = " + sampleRate
            + "\nDataLength = " + dataLength
            + "\nSamples = " + samples
            + "\ndata = " + data00 + " " + data01 + " " + data02 + " " +
                        data03 + " " + data04 + " " + data05 + " " +
                        data06 + " " + data07 + " " + data08 + " " +
                        data09 + " " + data10;

 return result ;
 }


} // end of class signalPdu.java
```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX C. DIS-Java-VRML DTD

```
<!--
             DIS-Java-VRML 1.2 Document Type Definition (DTD)

 Status:     Supports the following Protocol Data Units (PDUs):
             - EspduTransform (Entity State, Collision, Detonation, Fire)
             - ReceiverPdu, SignalPdu, Transmitter
             Testing results satisfactory.  Compact form also needed.

 Authors:    Don Brutzman and Dave Laflam

 Address:    http://www.web3D.org/TaskGroups/x3d/translation/DisJavaVrml.dtd

 References: http://www.web3D.org/WorkingGroups/vrtp/dis-java-vrml
             http://www.web3D.org/TaskGroups/x3d/translation/x3d-compromise.dtd

 Summary:    Define the DIS-Java-VRML tag and attribute profile for X3D.

             Elements in this tagset are enabled by setting the
             DIS-Java-VRMLProfile entity to "INCLUDE" at the top of an
             X3D scene file, as shown in several examples.  This flag then
             triggers inclusion of the DIS-Java-VRML tags in the
             x3d-compromise.dtd tagset.

 Created:    30 July 2000

 Revised:    25 September 2000
-->


<!-- Other node and attribute types (%SFBool; etc.) are defined in x3d-
compromise.dtd -->

<!ENTITY % RadioPduTypes   "(ReceiverPdu|SignalPdu|TransmitterPdu)">     <!--
Boolean      -->


<!-- No new or additional wrapper tag definitions needed.  Compact version of
this
     DisJavaVrml.dtd will be more concise and flexible by not using wrapper tags
-->


<!ELEMENT EspduTransform (children)? >
<!ATTLIST EspduTransform

       marking                   %SFString;   #IMPLIED

       readInterval              %SFTime;     "0.1"
       writeInterval             %SFTime;     "0"
       siteID                    %SFInt32;    #IMPLIED
       applicationID             %SFInt32;    #IMPLIED
       entityID                  %SFInt32;    #IMPLIED

       translation               %SFVec3f;    "0 0 0"
       rotation                  %SFRotation; "0 0 1 0"
       scale                     %SFVec3f;    "1 1 1"
       scaleOrientation          %SFRotation; "0 0 1 0"
       bboxCenter                %SFVec3f;    "0 0 0"
       bboxSize                  %SFVec3f;    "-1 -1 -1"
       center                    %SFVec3f;    "0 0 0"

       address                   %SFString;   #IMPLIED
```

```
      port                        %SFInt32;     #IMPLIED
      multicastRelayHost          %SFString;    #IMPLIED
      multicastRelayPort          %SFInt32;     #IMPLIED
      rtpHeaderExpected           %SFBool;      "false"

      active                      %SFBool;      "false"
      timestamp                   %SFTime;      #IMPLIED
      rtpHeaderHeard              %SFBool;      #IMPLIED

      collided                    %SFBool;      "false"
      collideTime                       %SFTime;      #IMPLIED
      detonated                   %SFBool;      "false"
      detonateTime                      %SFTime;      #IMPLIED
      fired1                      %SFBool;      "false"
      fired2                      %SFBool;      "false"
      firedTime                   %SFTime;      #IMPLIED
      munitionStartPoint          %SFVec3f;     #IMPLIED
      munitionEndPoint            %SFVec3f;     #IMPLIED

      articulationParameterCount        %SFInt32;     #IMPLIED
      articulationParameterValue0       %SFFloat;     #IMPLIED
      articulationParameterValue1       %SFFloat;     #IMPLIED
      articulationParameterValue2       %SFFloat;     #IMPLIED
      articulationParameterValue3       %SFFloat;     #IMPLIED
      articulationParameterValue4       %SFFloat;     #IMPLIED
      articulationParameterValue5       %SFFloat;     #IMPLIED
      articulationParameterValue6       %SFFloat;     #IMPLIED
      articulationParameterValue7       %SFFloat;     #IMPLIED
      articulationParameterValue8       %SFFloat;     #IMPLIED
      articulationParameterValue9       %SFFloat;     #IMPLIED
      articulationParameterValue10      %SFFloat;     #IMPLIED
      articulationParameterValue11      %SFFloat;     #IMPLIED
      articulationParameterValue12      %SFFloat;     #IMPLIED
      articulationParameterValue13      %SFFloat;     #IMPLIED
      articulationParameterValue14      %SFFloat;     #IMPLIED

      traceColor                  %SFColor;     #IMPLIED
      traceOffset                       %SFVec3f;     #IMPLIED
      traceSize                   %SFVec3f;     #IMPLIED
      traceJava                   %SFBool;      "false"

      nodeTypeHint                NMTOKEN             #FIXED "Transform"
      DEF                         ID            #IMPLIED
      USE                         IDREF         #IMPLIED >

<!-- also included in EspduTransform:  CollisionPdu DetonatePdu FirePdu -->

<!ELEMENT ReceiverPdu EMPTY >
<!ATTLIST ReceiverPdu

      whichGeometry               %SFInt32;     #IMPLIED
      radioPduType                %RadioPduTypes;     #IMPLIED

      readInterval                %SFTime;      "1"
      writeInterval               %SFTime;      "0"
      siteID                      %SFInt32;     #IMPLIED
      applicationID               %SFInt32;     #IMPLIED
      entityID                    %SFInt32;     #IMPLIED

      address                     %SFString;    #IMPLIED
      port                        %SFInt32;     #IMPLIED
      multicastRelayHost          %SFString;    #IMPLIED
      multicastRelayPort          %SFInt32;     #IMPLIED
      rtpHeaderExpected           %SFBool;      "false"

      active                      %SFBool;      "false"
      timestamp                   %SFTime;      #IMPLIED
      rtpHeaderHeard              %SFBool;      #IMPLIED
```

```
        radioID                     %SFInt32;     #IMPLIED
        receiverPower               %SFFloat;     #IMPLIED
        receiverState                       %SFInt32;     #IMPLIED
        transmitterSiteID                   %SFInt32;     #IMPLIED
        transmitterApplicationID            %SFInt32;     #IMPLIED
        transmitterEntityID                 %SFInt32;     #IMPLIED
        transmitterRadioID                  %SFInt32;     #IMPLIED

        traceJava                           %SFBool;      "false"
        nodeTypeHint                        NMTOKEN             #FIXED "Switch"
        DEF                                 ID            #IMPLIED
        USE                                 IDREF         #IMPLIED >


<!ELEMENT SignalPdu EMPTY >
<!ATTLIST SignalPdu

        whichGeometry                       %SFInt32;     #IMPLIED
        radioPduType                %RadioPduTypes;     #IMPLIED

        readInterval                        %SFTime;      "1"
        writeInterval                       %SFTime;      "0"
        siteID                              %SFInt32;     #IMPLIED
        applicationID                       %SFInt32;     #IMPLIED
        entityID                            %SFInt32;     #IMPLIED

        address                             %SFString;    #IMPLIED
        port                                %SFInt32;     #IMPLIED
        multicastRelayHost                  %SFString;    #IMPLIED
        multicastRelayPort                  %SFInt32;     #IMPLIED
        rtpHeaderExpected                   %SFBool;      "false"

        active                              %SFBool;      "false"
        timestamp                           %SFTime;      #IMPLIED
        rtpHeaderHeard                      %SFBool;      #IMPLIED

        radioID                             %SFInt32;     #IMPLIED
        encodingScheme                      %SFInt32;     #IMPLIED
        tdlType                             %SFInt32;     #IMPLIED
        sampleRate                          %SFInt32;     #IMPLIED
        samples                             %SFInt32;     #IMPLIED
        dataLength                          %SFInt32;     #IMPLIED
        data00                              %SFInt32;     #IMPLIED
        data01                              %SFInt32;     #IMPLIED
        data02                              %SFInt32;     #IMPLIED
        data03                              %SFInt32;     #IMPLIED
        data04                              %SFInt32;     #IMPLIED
        data05                              %SFInt32;     #IMPLIED
        data06                              %SFInt32;     #IMPLIED
        data07                              %SFInt32;     #IMPLIED
        data08                              %SFInt32;     #IMPLIED
        data09                              %SFInt32;     #IMPLIED
        data10                              %SFInt32;     #IMPLIED

        traceJava                                 %SFBool;      "false"
        nodeTypeHint                              NMTOKEN       #FIXED "Switch"
        DEF                                       ID            #IMPLIED
        USE                                       IDREF         #IMPLIED >


<!ELEMENT TransmitterPdu EMPTY >
<!ATTLIST TransmitterPdu

        whichGeometry                             %SFInt32;     #IMPLIED
        radioPduType                              %RadioPduTypes;     #IMPLIED

        readInterval                              %SFTime;      "1"
```

```
writeInterval                          %SFTime;     "0"
siteID                                 %SFInt32;    #IMPLIED
applicationID                          %SFInt32;    #IMPLIED
entityID                               %SFInt32;    #IMPLIED

address                                %SFString;   #IMPLIED
port                                   %SFInt32;    #IMPLIED
multicastRelayHost                     %SFString;   #IMPLIED
multicastRelayPort                     %SFInt32;    #IMPLIED
rtpHeaderExpected                      %SFBool;     "false"

active                                 %SFBool;     "false"
timestamp                              %SFTime;     #IMPLIED
rtpHeaderHeard                         %SFBool;     #IMPLIED

radioID                                %SFInt32;    #IMPLIED
antennaLocation                        %SFVec3f;    #IMPLIED
antennaPatternLength                   %SFInt32;    #IMPLIED
antennaPatternType                     %SFInt32;    #IMPLIED
cryptoKeyId                            %SFInt32;    #IMPLIED
cryptoSytem                            %SFInt32;    #IMPLIED
frequency                              %SFInt32;    #IMPLIED
inputSource                            %SFInt32;    #IMPLIED
lengthOfModulationParameters           %SFInt32;    #IMPLIED
modulationTypeDetail                   %SFInt32;    #IMPLIED
modulationTypeMajor                    %SFInt32;    #IMPLIED
modulationTypeSpreadSpectrum           %SFInt32;    #IMPLIED
modulationTypeSystem                   %SFInt32;    #IMPLIED
power                                  %SFInt32;    #IMPLIED
radioEntityTypeCategory                %SFInt32;    #IMPLIED
radioEntityTypeCountry                 %SFInt32;    #IMPLIED
radioEntityTypeDomain                  %SFInt32;    #IMPLIED
radioEntityTypeKind                    %SFInt32;    #IMPLIED
radioEntityTypeNomenclature            %SFInt32;    #IMPLIED
radioEntityTypeNomenclatureVersion     %SFInt32;    #IMPLIED
relativeAntennaLocation                %SFVec3f;    #IMPLIED
transmitFrequencyBandwidth             %SFInt32;    #IMPLIED
transmitState                          %SFInt32;    #IMPLIED

traceJava                              %SFBool;     "false"
nodeTypeHint                           NMTOKEN      #FIXED "Switch"
DEF                                    ID           #IMPLIED
USE                                    IDREF        #IMPLIED >
```

APPENDIX D. X3D PROTOS

Note: only the SHFAntennaPROTO.XML file is listed, the UHFAntennaPROTO.XML and RAUAntennaPROTO.XML file can be found in the DIS-Java-VRML distribution.

```
<X3D>
<Header>
  <meta
     name='AntennaWorld'
     content='AntennaWorld.xml'/>
  <meta
     name='author'
     content='Dave Laflam'/>
  <meta
     name='revised'
     content='15 July 2000'/>
  <meta
     name='description'
     content='SHFAntennaPROTO.XML'/>
  <meta
     name='url'
     content='enter url address here'/>
  <meta
     name='generator'
     content='X3D-Edit, http://www.web3D.org/
                        TaskGroups/x3d/translation/
                        README.X3D-Edit.html'/>
</Header>
<Scene>
  <!-- ExternProtoDeclare definitions must be included
       verbatim -->
  <ProtoDeclare
     name='SHFAntenna'>
       <field
          IS='ESPDU_TRANSFORM.translation'
          name='initialLocation'
          type='Vector3'
          value='0 0 0'/>
       <field
          IS='ESPDU_TRANSFORM.rotation'
          name='initialAzimuth'
          type='Rotation'
          value='0 0 1 0'/>
       <field
          IS='SHFAntennaPole.scale'
          name='antennaPoleScale'
          type='Vector3'
          value='1 1 1'/>
       <field
          IS='ESPDU_TRANSFORM.marking'
          name='marking'
          type='String'
          value='SHF antenna'
          vrml97Hint='field'/>
       <!--  -->
       <field
          IS='ESPDU_TRANSFORM.address'
          name='address'
          type='String'
          value='224.2.181.145'
          vrml97Hint='field'/>
       <field
          IS='ESPDU_TRANSFORM.port'
```

```
            name='port'
            type='Integer'
            value='62040'
            vrml97Hint='field'/>
         <field
            IS='ESPDU_TRANSFORM.siteID'
            name='siteID'
            type='Integer'
            value='0'
            vrml97Hint='field'/>
         <field
            IS='ESPDU_TRANSFORM.applicationID'
            name='applicationID'
            type='Integer'
            value='1'
            vrml97Hint='field'/>
         <!--  -->
         <field
            IS='ESPDU_TRANSFORM.entityID'
            name='entityID'
            type='Integer'
            value='0'
            vrml97Hint='field'/>
<LOD
  range='40000'>
    <level>
<!-- First child is rendered in range of 4000 -->
<EspduTransform
DEF='ESPDU_TRANSFORM'
readInterval='.25'
traceColor='0 0.51 0.06'
traceOffset='0 24 0 '
traceSize='10 10 10'>
<children>
   <Transform>
        <children>
             <Inline
                url='"SHFAntennaPole.wrl"'/>
             <Anchor
                description='UHF Antenna Site image'
                parameter='target=_blank'
                url='UHFAntennaSite.png'>
                  <children>
                       <Inline
                          url='"SHFAntennaDish.wrl"'/>
                  </children>
             </Anchor>
        </children>
   </Transform>
   <Transform
      DEF='DomeGreen4'
      scale='100 100 40'
      translation='60 0 0'>
        <children>
             <Inline
                DEF='Dome'
                url='"DomeGreen.wrl"'/>
        </children>
   </Transform>
   <Viewpoint
      description='SHF antenna side view'
      orientation='0 1 0 1.5708 '
      position='40 10 0'/>
</children>
 </EspduTransform>
 <WorldInfo/>
</level>
</LOD>
```

```
<Text
  string='SHF Text'/>
<!-- Script nodes will go here for
       ReceiverPdu, SignalPdu, TransmitterPdu.  Data values will get
       ROUTEd into (and out of) the antenna and signal visualization
     geometry. -->
</ProtoDeclare>
<!-- The following Instance allows the
Proto to be viewed when viewed as a standalone
file. -->
<!-- This is like paramaterization of the       UHF Antenna
 -->
<ProtoInstance
name='SHFAntenna'>
<fieldValue
  name='initialLocation'
  value='200 0 200'/>
<fieldValue
  name='entityID'
  value='70'/>
</ProtoInstance>
<WorldInfo
     info='"Authors:   David Laflam"
  "Revised:   12 July 00"
  "Purpose:   Brings SHF, UHF and Rau Antennas into project world"
  "Script:    none"
  "Browser:   CosmoPlayer"'
     title='AntennaWorld'/>
  <Background
     groundAngle='1.57079'
     groundColor='1 0.8 0.6, 0.6 0.4 0.2'
     skyAngle='0.2'
     skyColor='1 1 1, 0.2 0.2 1'/>
  <NavigationInfo
     speed='20'
     type='"EXAMINE" "ANY"'/>
  <Viewpoint
     DEF='High_Above_Airfields'
     description='High_Above_Airfields '
     fieldOfView='.7853'
     orientation='-.9996220469474792, -0.020204812288284302,
                  -0.01863904483616352, .35458293557167053'
     position='-2500, 1500, 6000'/>
</Scene>
</X3D>

<!-- Tag color codes:
<field> <NodeName attribute='value'/> </field> -->
```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX E. CD-ROM

Theses Appendix Published
as Part of the
Distributed Interactive Simulation
DIS-Java-VRML Working Group



X3D and DIS-Java-VRML Theses

MAJ David W. Laflam
US Army

Title:3D Visualization of Theater-Level Radio Communications Using a Networked Virtual Environment

ABSTRACT

The military is heavily reliant on the transfer of information among various networks in day-to-day operations. Radio-based communications networks that support this volume of information are complex, difficult to manage, and change frequently. Communications network planners need a way to clearly visualize and communicate mobile operational network capabilities, particularly to network users.

By using the DIS-Java-VRML simulation and modeling toolkit, visualizations of radio-frequency energy and radio path-profiling data can be quickly generated as 3D models. These animated 3D visualizations can be loaded into a networked virtual environment, so that communications planners can detect a variety of problems such as radio frequency interference and gaps in coverage. Planners can also brief senior staff, plan within their own staff, and collaborate with communications staff planners in distant locations using such virtual environments.

DIS-Java-VRML visualization tools can provide a clear picture of the battle space with respect to the deployed communications architecture. The prototypes presented in this thesis demonstrate the ability to generate a shared visualization that can show a radio communications network in 3D. Such dynamic visualizations increase communications planning information bandwidth and yield more intuitive ways of presenting information to users. Higher information density in a more intuitive format

enables better understanding with quicker reaction times. This thesis and the visualization tool discussed provide the foundation for fundamental improvements in visualizing radio communications environments.

Thesis Link
Software Reference Link

MAJ Thomas E. Miller
US Army
Integrating Realistic Human Group Behaviors Into A Networked 3d Virtual Environment
ABSTRACT

Virtual humans operating inside large-scale virtual environments (VE) are typically controlled as single entities. Coordination of group activity and movement is usually the responsibility of their "real world" human controllers. Georeferencing coordinate systems, single-precision versus double-precision number representation and network delay requirements make group operations difficult. Mounting multiple humans inside shared or single vehicles, (i.e. air-assault operations, mechanized infantry operations, or small boat/riverine operations) with high fidelity is often impossible.

The approach taken in this thesis is to reengineer the DIS-Java-VRML Capture the Flag game geolocated at Fort Irwin, California to allow the inclusion of human entities. Human operators are given the capability of aggregating or mounting nonhuman entities for coordinated actions. Additionally, rapid content creation of human entities is addressed through the development of a native tag set for the Humanoid Animation (H-Anim) 1.1 Specification in Extensible 3D (X3D). Conventions are demonstrated for integrating the DIS-Java-VRML and H-Anim draft standards using either VRML97 or X3D encodings.

The result of this work is an interface to aggregate and control articulated humans using an existing model with a standardized motion library in a networked virtual environment. Virtual human avatars can be mounted and unmounted from aggregation entities. Simple demonstration examples show coordinated tactical maneuver among multiple humans with and without vehicles. Live 3D visualization of animated humanoids on realistic terrain is then portrayed inside freely available web browsers.

Thesis Link
Software Reference Link

MAJ Mark Murray
MAJ Jason Quigley
US Air Force

Automatically Generating A Distributed 3d Battlespace Using Usmtf And Xml-Mtf Air Tasking Order, Extensible Markup Language (Xml) And Virtual Reality Modeling Language (Vrml)

ABSTRACT

For the past three decades, the Department of Defense (DoD) has used the U.S. Message Text Format (USMTF) as the primary means to exchange information and to achieve interoperability between joint and coalition forces. To more effectively exchange and share data, the Defense Information Systems Agency (DISA), the lead agency for the USMTF, is actively engaged in extending the USMTF standard with a new data sharing technology called Extensible Markup Language (XML). This work translates and synthesizes Air Tasking Order (ATO) data messages written in XML into a three-dimensional (3D) air attack plan within a virtual environment through the use of the Virtual Reality Modeling Language (VRML).

Thesis Link
Software Reference Link

X3D and DIS-Java-VRML Software Tools

A. X3DEdit
B. X3DEdit Examples
C.JDK.1.2.2
D. Netscape 4.73
E. COSMO Player(VRML Plug-in)
F. KELP Forest
G. VRTP Distribution
H. RRA(Recursive Ray Acoustic)
I. Xeena 1.2
J. Vorlon(VRML Syntax Checker)
K. Cygwin (Windows Unix Tools)
L. DIS-Java-VRML
M. Xj3D
N. GeoVRML

A. X3DEdit

X3D-Edit is a graphics file editor for Extensible 3D (X3D) that enables simple error-free editing, authoring and validation of X3D or VRML scene-graph files.

Local URL: www.web3D.org/TaskGroups/x3d/translation/README.X3D-Edit.html

Local Fie Download:X3D-Edit.zip

URL: http://www.web3d.org/x3d.html

B. X3DEdit Examples

Local Fie Download:X3D-Examples.zip

These are the examples for X3D-Edit based on VRML 2.0 Sourcebook by David Nadue

---

C. JDK.1.2.2

The essential Java 2 SDK, tools, runtimes, and APIs for developers writing, deploying, and running applets and applications in the Java programming language. Also includes earlier Java Development Kit versions JDKTM 1.1 and JRE 1.1

Local Fie Download: j2sdk1_3_0-win.exe

URL: http://java.sun.com/products/index.html

D. Netscape 4.73

Communicator 4.7 is the latest release of the Internet software suite from Netscape. In addition to the Netscape Navigator browser, Communicator includes a complete set of tools for effective everyday communication.

Local Fie Download: netscape-cc32d475.exe

URL: http://home.netscape.com/computing/download/index.html

E. COSMO Player (VRML Plug-in)

Cosmo Player is a high-performance, cross-platform VRML 2.0 client designed for fast and efficient viewing of virtual worlds. Navigate and manipulate 3D scenes and bring your Web experience to a new level.

Use VRML to fly through anatomy class, experience 3D data visualizations, or show off a CAD model. Cosmo Player is the premiere viewing client for VRML, with support for the latest standards.

Whether on the Internet or in an enterprise, Cosmo Player allows web content creators and applications developers to add visual and multimedia elements to their work.

Local Fie Download: cosmo_win95nt_eng.exe

URL: http://www.cai.com/cosmo/html/win95nt.htm

F. KELP Forest

Two classes of graduate students learning 3D graphics and analytic simulation at the Naval Postgraduate School modeled the three-dimensional (3D) shape, structure, imagery and motion behaviors of plants and animals in the Kelp Forest Exhibit at the Monterey Bay Aquarium. Our intended audience includes educators and students of all ages, scientific users interested in composing models in a 3D Web environment, and the general public. By focusing on thoroughly modeling a controlled environment, we produced an exemplar 3D graphics site for modeling larger and more sophisticated underwater domains. The Virtual Reality Modeling Language (VRML) proved to be an excellent medium for capturing diverse models, composing multiple student efforts, and publishing dynamic results publicly on the Web. This project was successfully demonstrated to 1000 people during the National Ocean Fair in Monterey June 12 1998.

Local URL: kelp/index.html

Local Fie Download: kelp.zip

URL: http://web.nps.navy.mil/~brutzman/kelp

G. VRTP Distribution

The capabilities of the Virtual Reality Modeling Language (VRML) permit building large-scale virtual environments using the Internet and the World Wide Web. However the underlying network support provided by the hypertext transfer protocol (http) is insufficient for large-scale virtual environments. Additional capabilities for many-to-many peer-to-peer communications plus network monitoring need to be combined with the client-server capabilities of http. To accomplish this task, we present a detailed design rationale for the virtual reality transfer protocol (vrtp). vrtp is designed to support interlinked VRML worlds in the same manner as http was designed to support interlinked HTML pages. vrtp will be optimized in two ways: on individual desktops and across the Internet. vrtp appears to be a necessary next step in the deployment of all-encompassing interactive internetworked 3D worlds.

Local URL: vrtp/vrtp/index.html

Local Fie Download: vrtp.zip

URL: http://www.web3d.org/WorkingGroups/vrtp/

H. RRA (Recursive Ray Acoustic)

This project calculates and renders physically realistic sonar beams in real time. Java programs are used for sonar ray-tracing computation and the Virtual Reality Modeling Language (VRML 97) is used for 3D graphics.

The primary motivation for this project is to produce underwater sonar beams for analytic and visualization use in virtual worlds. Virtual world simulations are realistic when individual components are simulated in a manner that reflects reality. For an underwater virtual world that includes simulated acoustic detection, a physically based sonar propagation model is required if ranges in excess of tens of meters are expected. The Recursive Ray Acoustics (RRA) Algorithm by Dr. Lawrence Ziomek of NPS provides a general & rapid ray-tracing algorithm which can accurately & quickly predict sonar propagation through seawater, under a wide variety of surface, water-column and ocean-bottom environmental conditions.

This project creates an application programming interface (API) for real-time 3D computation and visualization of acoustic energy propagation. The API provides features for generating complex physically based sonar information at interaction rates, and then visualizing that acoustic information. The simulation is programmed in Java, and runs either as a stand-alone program or as a script in a web browser. This program generates Virtual Reality Modeling Language (VRML 97) compliant code that can be viewed from any VRML-capable Web browser. This approach allows the characteristics of the energy propagation to be calculated with high precision and observed in three dimensions (3D) and in real time.

As sonar-system information bandwidth becomes larger, more intuitive ways of presenting information to users are required. Interactive 3D graphics with environmental and entity rendering can free users from from mentally integrating complex data piecemeal. This approach can enable significantly greater understanding and quicker reaction times. We are optimistic that this API might someday provide the foundation for fundamental advances in sonar modeling and visualization.

Local URL: rra/vrtp/rra/rra.html

Local Fie Download: rra.zip

URL: http://web.nps.navy.mil/~brutzman/vrtp/rra/rra.html

I. Xeena 1.2

Xeena is a generic Java application from the IBM Haifa Research Laboratory for editing valid XML documents derived from any valid DTD. The editor takes as input a given DTD, and automatically builds a palette containing the elements defined in the DTD. Users can thus create/edit/expand any document derived from that DTD, by using a

visual tree-directed paradigm. The visual paradigm requires a minimum learning curve as only valid constructs/elements are presented to the user in a context-sensitive palette. A Key feature of Xeena is its syntax directed editing ability. Xeena is aware of the DTD grammar, and by making only authorized elements icons sensitive, automatically ensures that all documents generated are valid according to the given DTD.

Local Fie Download: Xeena-1.2EA.exe

URL: http://www.alphaworks.ibm.com/tech/xeena

J. Vorlon(VRML Syntax Checker)

The industry standard command line VRML Validator
Trapezium developed Vorlon as a service to the VRML community to allow authors to spend less time chasing bugs and more time creating high quality content.
Validates conformance to VRML97 specification
Displays line, line number and descriptive message
for each error or warning
Vorlon is Freeware.

Local Fie Download: vorlon.exe

URL: http://www.trapezium.com/VorlonPage.htm

K. Cygwin (Windows Unix Tools)

Cygwin brings a standard UNIX/Linux shell environment, including many of its most useful commands, to the Windows platform so IT managers can effectively deploy trained staff, and leverage existing investments in UNIX/Linux source code and shell scripts.

URL: http://www.cygnus.com/cygwin/

L. DIS-Java-VRML

The area of interest of this working group is the nexus of DIS, Java and VRML. The IEEE Distributed Interactive Simulation (DIS) Protocol is used to communicate state information (such as position, orientation, velocities and accelerations) among multiple entities participating in a shared network environment. Java is a portable networked programming language that can interoperate on any computer which includes a Web browser. The Virtual Reality Modeling Language (VRML) enables platform-independent interactive three-dimensional (3D) graphics across the Internet, and can be used to compose sophisticated 3D virtual environments.

The DIS-Java-VRML Working Group is developing a free software library, written in Java and interoperable with both DIS and VRML. There are a number of people contributing to the public-domain code archive. This software is protected under the terms of the GNU General Public License.

Local URL: vrtp/dis-java-vrml/index.html

Local Fie Download:dis-java-vrml.zip

URL: http://www.web3d.org/WorkingGroups/vrtp/dis-java-vrml/

M. Xj3D

Xj3D is an example implementation for the X3D specification. Specifically, Xj3D is a Java3D-based open-source loader, browser and exporter for Extensible 3D (X3D) graphics.

URL: http://web3d.metrolink.com/cgi-bin/cvsweb.cgi/x3d/HowToInstall.html

---

N. GeoVRML

GeoVRML is an official Working Group of the Web3D Consortium. It was formed on 27 Feb 1998 with the goal of developing tools and recommended practice for the representation of geographical data using the Virtual Reality Modeling Language (VRML). The desire is to enable geo-referenced data, such as maps and 3-D terrain models, to be viewed over the web by a user with a standard VRML plugin for their web browser.

Local URL: GeoVRML/1.0/doc/index.html

Local Fie Download: geovrml1_0.exe

URL: http://www.ai.sri.com/geovrml/

September 24 2000 (official NPS disclaimer)
URL: www.web3D.org/WorkingGroups/vrtp/dis-java-vrml/SoftwareReference.html
feedback: feedback@nps.navy.mil

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF REFERENCES

Abernathy, M. and Shaw, S., (1998) "*Integrating Geographic Information in VRML Worlds*," Proceedings VRML 98: Third Symposium of the Virtual Reality Modeling Language, Monterey, California, February 16-19, pp. 107-114. Available at:http://ece.uwaterloo.ca/vrml98/cdrom/papers/abernath/abernath.pdf

Ames, A. L., Nadeau, D. R., Moreland, J. L., (1997) *VRML 2.0 Sourcebook 2nd* Edition , Littleton, Massachusetts, USA, John Wiley & Sons, Available at: http://www.wiley.com/compbooks/vrml2sbk/cover/cover.htm

Brutzman, D. P., (1994*), A Virtual World for an Autonomous Underwater Vehicle*, PhD Thesis, Naval Postgraduate School, Monterey, California, USA. Available at: http://web.nps.navy.mil/~brutzman/dissertation

Brutzman, Don, (1998) "*The Virtual Reality Modeling Language and Java*," Communications of the ACM, vol. 41 no. 6, June, pp. 57-64. Available at: http://web.nps.navy.mil/~brutzman/vrml/vrmljava.pdf

Canterbury, M. (1995), *An Automated Approach to Distributed Interactive Simulation (DIS) Protocol Development*, Master's Thesis, Naval Postgraduate School, Monterey, California, USA. Available at: http://www.npsnet.org/~zyda/Theses/Michael.Canterbury.pdf

Carr J. J., (1998), *Practical Antenna Handbook*,3rd Edition, New York, NY, McGraw-Hill Publishing.

COMNET III/STK – (2000) Available at: http://www.stk.com and http://www.caci.com

Defanti, Thomas, (1987)"*Insight Through Images*," A Unix Review, Vol. 7, No. 3.

Deitel H. M., Deitel P. J. (1999), *Java™ How to Program*, Upper Saddle River, New Jersey, Prentice Hall, INC. Available at: http://www.deitel.com

Eick, Stephen G.,"Visual Discovery and Analysis", (2000) *IEEE Transactions on Visualization and Computer Graphics*, January-March, Volume 6 Number 1 Pg. 44.

Fairborn, David, and Parsley, Scott, (1997) "*The Use of VRML for Cartographic Presentation,"* Computers and Geosciences, vol. 23, no. 4, May, pp 475-481.

FGDC - Federal Geographic Data Committee, (1994) "FGDC Content Standard for Digital Metadata," June 8, Available at: http://fgdc.er.usgs.gov

Hess, G. C., (1998), Handbook of Land Mobile Radio System Coverage. Norwood, Massachusetts, USA, Artech House, INC. Available at: http://www.techbooks.co.uk/artech/book26.htm

Iverson, Lee, "GeoVRML Working Group Home Page," July 23, 1999. Available at http://www.ai.sri.com/~leei/geovrml

Jian R., (1991), *The Art of Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling, Littleton*, Massachusetts, USA, John Wiley & Sons, INC.

Karahalios, Margarida, *Underwater Source Localization Using Scientific Data Visualization*, Masters Thesis,Department of Computer Science and Engineering, University of South Florida, July 1991.

Keller, Peter r., Keller, M. M., (1993), *Visual Cues Practical Data Visualization*, Los Alamitos, CA IEEE Computer Society Press.

Ladd, S. R.,(1988) *Java Algorithms*, New York, NY, McGraw-Hill Publishing.

Lea R., Matsuda K., Miyashita K.,(1996) *Java™ for 3D and VRML Worlds*, Indianapolis, Indian, New Riders Publishing.

Moscardini , A. O., Robson, E. H., (1988). *Mathematical Modeling for Information Technology: Telecommunications Reception Transmission, and Security*. Chichester, West Sussex, England, Ellis Horwood Limited.

McCullagh, Michael J., (1997) "*Quality, Visualization, and Use of Terrain Models in Physical System*" Available at: http://ncgia.ncgia.ucsb.edu/conf/SANTA_FE_CD-ROM

Mobile Subscriber Equipment Network Planning Terminal (MSE-NPT) v2.03, (1997) TB-11-5895-1544-10-2, Headquarter, Department of the Army.

OPNET - 2000 Available at: http://www.opnet.com/opnet/about.html and http://web.singnet.com.sg/~meridian/products

Powell, Colin L., (1995), Joint Pub 6-0, *Doctrine for Command, Control, Communications, and Computers (C4) Systems Support to Joint Operation.*

Roehl, B., Couch, J., Reed-Ballriech, C., Rohaly, T., Brown, G.,(1997), Late Night VRML 2.0 with Java™ , Emeryville, California, Macmillan Computer Publishing. Available at: http://ece.uwaterloo.ca/~broehl/vrml/lnvj

RIS - Rapid Imaging Software Home Page, 3D Terrain Modeling Tool, May 24, 1998. Available at http://www.landform.com/vrml.htm

Sandeep, S., and Zyda, M.,(1999.) *Networked Virtual Environments Design and Implementation* , ACM Press, New York, NY. Available at: http://www.npsnet.org/~zyda/Books.html

Silicon Graphics, Cosmo Player Version 2.1 VRML Browser, 1998. Available at: http://www.cai.com/cosmo

Sowizral, Henry A. and Deering, Michael F., "The Java3D API and Virtual Reality", *IEEE Computer Graphics and Applications* (May/June 1999).

Sowizral, H., Nadaeu, D., Anderson, D., Bailey, M., Deering, M. "Introduction to Programming with Java3D," ACM SIGGRAPH 98, Course #37 Course Notes, July 1998, in form of a printed book and on the course CD-ROM.

Weiss, Mark. A., (1999) Data Structures and Problem Solving Using Java™, Menlo Park, California, Addison Wesley Logman, INC.

VRML -Virtual Reality Modeling Language, International Standard ISO/IEC 14772-1:1997. Available at: http://www.web3d.org/technicalinfo/specifications/vrml97/index.htm

XML - Extensible Markup Language, REC-xml-19980210 W3C Recommendation, 10-February-1998. Available a*t:* http://www.w3.org/TR/REC-xml.html Weiss, Mark. A., (1999) Data Structures and Problem Solving Using Java™, Menlo Park, California, Addison Wesley Logman, INC.

THIS PAGE INTENTIONALLY LEFT BLANK

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center 2
   8725 John J. Kingman Road, Ste 0944
   Ft. Belvoir, Virginia 22060-6218

2. Dudley Knox Library 2
   Naval Postgraduate School
   411 Dyer Rd.
   Monterey, California 93943-5101

3. Robert J. Barton III 1
   Fraunhofer Center for Research in Computer Graphics (CRCG)
   321 South Main St
   Providence, RI 02903

4. Dr. Michael P. Bailey 1
   Technical Director, Marine Corps Training and Education Command
   Commanding General
   Marine Corps Combat Development Command, Code 46T
   3300 Russell Road
   Quantico, VA 22134

5. Dr. Philip S. Barry 1
   Chief, S&T Initiatives Division
   Defense Modeling and Simulation Office
   1901 N. Beauregard Street, Suite 500
   Alexandria VA 22311

6. Curtis Blais 1
   Institute for Joint Warfare Analysis
   Naval Postgraduate School

7. Don Brutzman, Code UW/Br 1
   Naval Postgraduate School
   Monterey, CA 93940-5000

8. Rex Buddenberg Code IS/Bu 1
   Naval Postgraduate School
   Monterey, CA 93940-5000

9. Research Assistant Professor Michael Capps, CodeCS/Cm 1
   Computer Science Department
   Naval Postgraduate School
   Monterey, CA 93940-5000

10. Capt. Steve Chapman, USN 1
    N6M
    2000 Navy Pentagon
    Room 4C445
    Washington, DC 20350-2000

11. Erik Chaum 1
    NAVSEA Undersea Warfare Center
    Division Newport
    Code 2231, Building 1171-3
    1176 Howell Street
    Newport, RI 02841-1708

12. COL K. Steven Collier USA 1
    Army Model and Simulation Office
    1111 Jefferson Davis Highway, Suite 503E
    Arlington, VA 22202

13. Dr. James Eagle 1
    Code UW
    Chair, Undersea Warfare Academic Group
    NPS

14. Dr. Paul Fishwick 1
    Computer & Information Science and Engineering Department
    University of Florida
    Post Office Box 116120
    322 Building CSE
    Gainesville, FL 32611-6120

15. Connell Gallagher 1
    President
    Parallel Graphics
    36 Upper Fitzwilliam Street
    Dublin 2, Ireland

16. CDR Arthur Galpin, USN 1
US Specail Operation Command (SORR-SCS)
7701 Tampa Point Boulevard
MacDill Air Force Base, FL 33621-5323

17. MR. Jerry Ham 1
National Simulation Center (NSC)
ATTN:ATZL-NSC
410 Kearney Avenue --- Building 45
Fort Leavenworth, KS 66027-1306

18. Dr. Tony Healey 1
Code ME/Hy
NPS

19. David Holland 1
Computer Science/C3I Center MS4A5
George Mason University
Fairfax, VA 22032

20. Pamela Krause 1
Advanced Systems & Technology
National Reconnaissance Office
14675 Lee Road
Chantilly Virginia 20151-1714

21. John Lademan 1
Electronic Sensors and Systems Sector
Northrop Grumman Corporation
PO Box 1488 - MS 9030
Annapolis MD 21404

22. CPT(P) David W. Laflam 1
173 Brook Rd.
Sanbornton NH 03269

23. Jaron Lanier 1
Advanced Network & Services, Inc.
200 Business Park Drive
Armonk NY 10504 USA

24. Dr. R. Bowen Loftin 1
Director of Simulation Programs
Virginia Modeling Analysis & Simulation Center
Old Dominion University
7000 College Drive
Suffolk VA 23435

25. CAPT Arnold O. Lotring USN 1
Commanding Officer
Naval Submarine School
Code 00, Naval Submarine School
PO Box 700
Groton, CT 06349-5700

26. Mike Macedonia 1
Chief Scientist and Technical Director
US Army STRICOM
12350 Research Parkway
Orlando, FL 32826-3276

27. Michael McCann 1
Monterey Bay Aquarium Research Institute (MBARI)
Post Office Box 628
Moss Landing, CA 95039-0628

28. CDR John C, Mickey, USN 1
PEO for Submarines (PMS401)
2531 Jefferson Davis Highway
NC3, Room 3W30
Arlington, VA 22242-5161
ATTN: CDR John Mickey

29. CAPT Bill Molloy USN 1
Chief Modeling & Simulation
Joint Warfighting Center
US Joint Forces Command
116 Lake View Parkway
Suffolk VA 23435-2697

30. Colonel (P) Dennis C. Moran 1
USCENTCOM/CC6
7115 South Boundary Blvd.
MacDill AFB, FL 33621-5101

31. CAPT Mark Murray USAF 1
Joint Battlespace Infosphere (JBI)
AFRL/IFSE
Building 3, Room E-1078
525 Brooks Road
Rome, NY 13441-4505

32. Michael Myjak 1
Vice President and CTO
The Virtual Workshop
P.O. Box 98
Titusville FL 32781

33. George Phillips 1
CNO, N6M1
2000 Navy Pentagon
Room 4C445
Washington, DC 20350-2000

34. Dr. Mark Pullen 1
Department of Computer Science/C3I Center MS4A5
George Mason University
FairFax, VA 22030

35. CAPT Jason Quigley USAF 1
Joint Battlespace Infosphere (JBI)
AFRL/IFSE
Building 3, Room E-1078
525 Brooks Road
Rome, NY 13441-4505

36. Dr. Martin Reddy 1
SRI International, EK219
333 Ravenswood Avenue
Menlo Park, CA 94025

37. Bernie Roehol 1
68 Margaret Avenue North
Waterloo, Ontario
N2J3P7
Canada

38. Dr. R. Jay Roland, President 1
Rolands and Associates
500 Sloat Avenue
Monterey CA 93940

39. MAJ Glenn Roussos, USA 1
US Special Operation Command (SORR-SCS)
7701 Tampa Point Boulevard
MacDill Air Force Base, FL 33621-5323

40. Dr. Sandeep Singhal
ReefEdge, Inc. 1
96 Linwood Plaza, PMB 505
Fort Lee, NJ 07024-3701

41. Keith Victor 1
Virtok Technologies, Inc
551 Belle Meade Farm Drive
Loveland, OH 45140

42. CAPT Robert Voigt 1
Chair, Electrical Engineering Department
U.S. Naval Academy
Annapolis MD 21402

43. Walter H. Zimmers 1
Defense Threat Reduction Agency
CPOC
6801 Telegraph Road
Alexandria VA 22310-3398

44. Dr. Michael Zyda, CodeCS/Zk 1
Chair, Modeling Virtual Environments and Simulation Academic Group
Computer Science Department
Naval Postgraduate School
Monterey, CA 93940-5000