NAVAL POSTGRADUATE SCHOOL

MONTEREY CALIFORNIA

THESIS

**TACTICAL WEB SERVICES:
USING XML AND JAVA WEB SERVICES
TO CONDUCT REAL-TIME NET-CENTRIC
SONAR VISUALIZATION**

by

Scott Rosetti

September 2004

Thesis Advisor: Don Brutzman
Second Reader: Duane Davis

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE		*Form Approved OMB No. 0704-0188*

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.

1. AGENCY USE ONLY (*Leave blank*)	2. REPORT DATE September 2004	3. REPORT TYPE AND DATES COVERED Master's Thesis	
4. TITLE AND SUBTITLE: Tactical Web Services: Using XML and Java Web Services to Conduct Real-Time Net-Centric Sonar Visualization		**5. FUNDING NUMBERS**	
6. AUTHOR Rosetti, Scott A.			
7. PERFORMING ORGANIZATION NAME AND ADDRESS Naval Postgraduate School Monterey, CA 93943-5000		**8. PERFORMING ORGANIZATION REPORT NUMBER**	
9. SPONSORING /MONITORING AGENCY NAME AND ADDRESS NAVAIR Patuxent River, MD		**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

With the unveiling of ForceNet, the Navy's architectural framework for how naval warfare is to be conducted in the information age, much of the technological focus has been placed on Web technology. One of the most promising technologies is Web services. Web services provide for a standard way to move and share data more reliably, securely, and quickly. The capabilities imbedded in Extensible Markup Language (XML) and Simple Object Access Protocol (SOAP) can merge previously disparate systems into one integrated environment. Already proven successful in the administrative realm, wide-area networks such as the Secure Internet Protocol Network (SIPRNET) have become secure and reliable enough to pass data between systems and units to support tactical operations. The Modeling, Virtual Environments and Simulation (MOVES) Institute at the Naval Postgraduate School is currently working to extend these precepts into the modeling and simulation world under the Extensible Modeling and Simulation Framework (XMSF) project. By leveraging existing Web service technology, warfighters at the "tip of the spear" can have access to previously unrealized amounts of tactically-relevant data, analysis, and planning tools.

The goal of this thesis is to apply the XMSF and Extensible 3D (X3D) graphics to the field of sonar visualization. Undersea warfare is a complex operation that requires a continuous and detailed analysis of the acoustic environment. Tactical sensor employment without a firm understanding of the complete undersea picture can lead to fatal consequences. The Navy has spent significant resources to develop training systems and tactical decision aids in an effort to integrate training, rehearsal and execution. Unfortunately, many of the high-resolution analysis tools that can provide high-resolution sonar prediction results are not easily accessible to the fleet. By taking advantage of Web services and XMSF technology, warfighters will need only access to the network to be able to pull real-time environmental analysis data from large databases, remotely run sonar prediction models on supercomputers, and view detailed three-dimensional (3D) virtual worlds that visualize the undersea picture.

14. SUBJECT TERMS Web services, visualization, modeling, simulation, sonar, XML, X3D, Xj3D, RRA, XMSF, tactical supercomputing			15. NUMBER OF PAGES 249
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT Unclassified	**18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	**19. SECURITY CLASSIFICATION OF ABSTRACT** Unclassified	**20. LIMITATION OF ABSTRACT** UL

THIS PAGE INTENTIONALLY LEFT BLANK

This thesis done in cooperation with the MOVES Institute
Approved for public release: distribution is unlimited

TACTICAL WEB SERVICES:
USING XML AND JAVA WEB SERVICES TO
CONDUCT REAL-TIME NET-CENTRIC SONAR VISUALIZATION

Scott A. Rosetti
Lieutenant, United States Navy
B.A., University of Notre Dame, 1998

Submitted in partial fulfillment of the
requirements for the degree of

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

from the

NAVAL POSTGRADUATE SCHOOL

September 2004

Author: Scott A. Rosetti

Approved by: Dr. Don Brutzman
                 Thesis Advisor

                 CDR Duane Davis, USN
                 Second Reader

                 Dr. Rudy Darken
                 Chair, MOVES Academic Committee

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

With the unveiling of ForceNet, the Navy's architectural framework for how naval warfare is to be conducted in the information age, much of the technological focus has been placed on Web technology. One of the most promising technologies is Web services. Web services provide for a standard way to move and share data more reliably, securely, and quickly. The capabilities imbedded in Extensible Markup Language (XML) and Simple Object Access Protocol (SOAP) can merge previously disparate systems into one integrated environment. Already proven successful in the administrative realm, wide-area networks such as the Secure Internet Protocol Network (SIPRNET) have become secure and reliable enough to pass data between systems and units to support tactical operations. The Modeling, Virtual Environments and Simulation (MOVES) Institute at the Naval Postgraduate School is currently working to extend these precepts into the modeling and simulation world under the Extensible Modeling and Simulation Framework (XMSF) project. By leveraging existing Web service technology, warfighters at the "tip of the spear" can have access to previously unrealized amounts of tactically-relevant data, analysis, and planning tools.

The goal of this thesis is to apply the XMSF and Extensible 3D (X3D) graphics to the field of sonar visualization. Undersea warfare is a complex operation that requires a continuous and detailed analysis of the acoustic environment. Tactical sensor employment without a firm understanding of the complete undersea picture can lead to fatal consequences. The Navy has spent significant resources to develop training systems and tactical decision aids in an effort to integrate training, rehearsal and execution. Unfortunately, many of the high-resolution analysis tools that can provide high-resolution sonar prediction results are not easily accessible to the fleet. By taking advantage of Web services and XMSF technology, warfighters will need only access to the network to be able to pull real-time environmental analysis data from large databases, remotely run sonar prediction models on supercomputers, and view detailed three-dimensional (3D) virtual worlds that visualize the undersea picture.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

viii

LIST OF FIGURES

xiii

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF ACRONYMS AND ABBREVIATIONS

2D	Two-dimensional
3D	Three-dimensional
API	Application Programmers Interface
ASTRAL	Acoustic Transmission Loss
ATO	Air Tasking Order
AUV	Autonomous Underwater Vehicle
AVCL	Autonomous Vehicle Control Language
BT	Bathythermograph
C2	Command and Control
C2IEDM	Command and Control Information Data Exchange Model
CASS	Comprehensive Acoustic Simulation System
CBR	Chemical, Biological and Radiological
CNO	Chief of Naval Operations
CUP	Common Undersea Picture
CVS	Concurrent Versioning System
DoD	Department of Defense
DOM	Document Object Model
DSS	Decision Support System
ETOPO2	Global 2 Elevation
FEC	Forward Error Correction
FNC	Functional Namespace Coordinators
FNMOC	Fleet Numerical Oceanography and Meteorology

GIG	Global Information Grid
GRAB	Gaussian Ray Bundle
GUI	Graphical User Interface
HTML	Hyper-Text Markup Language
HTTP	Hyper-Text Transfer Protocol
HUD	Heads-Up Display
IMAT	Integrated Multisensor Acoustic Training
IP	Internet Protocol
IT	Information Technology
IWEDA	Integrated Weather Effects Decision Aid
JDOM	Java Document Object Model
JPG	Joint Photographic Experts Group
JTIDS	Joint Tactical Information Distribution System
JWSDP	Java Web Services Developer's Pack
LOB	Line of Bearing
LOB	Local Observation Broadcast
M&S	Modeling and Simulation
MBL	Metcast Brokering Language
METCAST	Meteorology Forecast
MODAS	Modular Ocean Data Assimilation System
MOVES	Modeling, Virtual Environments, and Simulation
MRS	Metcast Response Schema
MWS	Metcast Web Service
NAVAIR	Naval Air Warfare Center

NetCDF	Network Common Data Form
NIPRNET	Non-secure Internet Protocol Router Network
NOGAPS	Navy Operational Global Atmosphere Prediction System
NPS	Naval Postgraduate School
ONR	Office of Naval Research
OODA	Observe, Orient, Decide, Act
OTIS	Optimum Thermal Interpolation System
PC-IMAT	Integrated Multisensor Acoustic Training – Personal Computer
PDA	Personal Data Assistant
PE	Parabolic Equation
PNG	Portable Network Graphics
RIMPAC	Rim of the Pacific
RMI	Remote Method Invocation
RPC	Remote Procedure Call
RRA	Recursive Ray Acoustics
RRAS	Recursive Ray Acoustic Simulator
SAAJ	SOAP with Attachments API for Java
SAVAGE	Scenario Authoring and Visualization for Advanced Graphical Environments
SAWS	SAVAGE Archive Web Service
SAX	Simple API for XML
SGML	Standard Generalized Markup Language
SIPRNET	Secure Internet Protocol Router Network
SMS	Sonar Modeling Schema

SOAP	Simple Object Access Protocol
SSP	Sound Speed Profile
SSPS	Sonar Source Properties Schema
SVG	Scalable Vector Graphics
SVWP	Sonar Visualization Workbench Panel
TAO	Tactical Action Officer
TDA	Tactical Decision Aid
TFW	Task Force Web
UDDI	Universal Description, Discovery and Integration
USW	Undersea Warfare
WEAVER	Web-Enabled Acoustic Visualization for Evaluation and Research
WSDL	Web Services Definition Language
WW3	Wave Watch III
VRML	Virtual Reality Markup Language
X3D	Extensible 3D Graphics
XML	Extensible Markup Language
XMSF	Extensible Modeling and Simulation Framework
XSBC	XML Schema-based Binary Compression
XSLT	Extensible Stylesheet Language for Transformation
XTC	XML Tactical Chat

ACKNOWLEDGMENTS

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

A.   OVERVIEW

This thesis provides the overarching architecture and supporting application programmers interfaces (APIs) designed to leverage existing Web technologies to put high-performance sonar prediction capability into the hands of the warfighter. By adopting a net-centric approach, operators can possess the capability to realistically simulate, forecast, and visualize the undersea picture over the network and then easily share this information with other deployed units. The net-centric paradigm is appropriately applied to sonar visualization. High-performance supercomputing grids and large-scale databases are required to achieve the appropriate resolution to be tactically significant, yet are impractical to install onboard most naval air and sea platforms. By allowing deployed users to remotely access these resources over the internet, Web services can bring the power of high-resolution modeling and simulation to the deployed fleet.

ForceNet is the architectural framework that will enable the realization of *Sea Power 21*, the Chief of Naval Operations' (CNO) vision for Navy transformation. One of the main tenets of ForceNet is the development of a Web-enabled Navy. Task Force Web (TFW) was created in 2001 to oversee the implementation of a Web-service architecture that can share data quickly and reliably (Swallow and Barrett, 2003). Naval applications and databases, both administrative and operational, are being recoded as Web services. The increased reliability, security and bandwidth of Department of Defense (DoD) networks, specifically the Secret Internet Protocol Router Network (SIPRNET), have opened the door to expand this effort to tactical information. Already, SIPRNET-based Web sites and chat rooms have become the informal standard for tactical coordination between forward-deployed units. The quick adaptation of these systems by the warfighter has made it abundantly clear that net-centric systems will be the backbone of naval operations for the foreseeable future.

Extending Web services to tactical applications is the next logical step. One implementation of Web services allows users to remotely access computers on the

network using standard Hyper-Text Transfer Protocol (HTTP). In short, Web services can make massive tactical databases and high-performance supercomputing grids readily available to the warfighter over the SIPRNET. This information can even be ingested into tactical decision aids or eventually live combat systems. This sort of tactical information can be a difference maker in operational scenarios, and can be made readily accessible throughout the fleet using Web services. These "Tactical Web Services" are the next step in creating a truly net-centric force.

Sonar modeling is a great candidate for Web service implementation. Sonar prediction models have always been regarded as a valuable resource, but have not been traditionally available in a real-time tactical environment. High-performance super-computing grids are an important analytical resource, but also do not support tactical operations well. In the current operational environment, if an enemy submarine was reported by a deployed surface unit, there would be little time to coordinate with land-based sonar prediction assets while planning imminent search or tracking operations. The warfighter might be left to guess critical tactical parameters. Some platforms carry systems of moderate resolution, but these are not universally available throughout the fleet. This problem has been well-identified, and several programs are currently in motion designed to integrate undersea warfare resources. However, it helps the warfighter to possess an alternate resource that could provide sonar modeling capabilities that do not require extensive software or hardware maintenance, upkeep, and storage. Exposing undersea warfare (USW) resources via Web services can allow an operator to quickly access oceanographic analysis from servers such as those hosted at Fleet Numerical Meteorology and Oceanography Center (FnMOC), run high-resolution sonar prediction models on land-based supercomputing grids, and be able to examine highly-detailed, three-dimensional (3D) visualizations needing nothing more than a Web browser. It is this sort of net-centric capability that can truly make a difference in the fleet. Such capabilities can also reduce the potentially harmful side effects of high-power sonar usage in peacetime.

B.   MOTIVATION

It is readily accepted that real-time simulation of sonar performance combined with intuitive visualizations of that information can significantly enhance warfighter performance in tasks such as designing buoy fields, tuning equipment, and making tactical decisions. Many technologies exist today that can be used to provide oceanographic information at the resolution needed to make a considerable tactical impact. Unfortunately, experience has shown that these technologies are not typically available where they are needed most, in the hands of the warfighter. It is generally not feasible (nor desired) to reproduce high-cost, high-maintenance supercomputing grids on every ship and aircraft in the fleet. By leveraging existing web technologies, these high-performance legacy systems can be easily accessed either directly from a webpage or through a downloadable thin client interface. An operator can then conduct high-resolution sonar modeling in real time from any computer on the appropriate network. In a broader scope, supercomputers become a tactical asset to the deployed fleet. These are significant new capabilities.

C.   PURPOSE

The purpose of this thesis is to provide a prospective architecture to create a net-centric sonar visualization client/server relationship and application programming interfaces (API) to access, process, and visualize ocean environmental data. The architecture will consist of the organization, technologies and standards needed to create such a networked sonar visualization tool. Two separate APIs will provide methods that the TDA will invoke to access an environmental forecasting database and create three-dimensional (3D) visualizations based off sonar model outputs that conform to a specific schema. In addition, an exemplar tactical decision aid (TDA) module was created to allow for the demonstration of early work and to show another possible solution that avoids certain limitations inherent in a strict browser application. Extensible 3D (X3D) graphics and Extensible Markup Language (XML) data representations are used throughout to maximize Web-based XMSF capabilities.

D.   GOALS

The original focus of this thesis was to develop real-time data visualization techniques to enhance tactical undersea warfare tactical decision aids. As part of a project to develop a prototype net-centric decision aid, the attention of this research expanded to leveraging existing Web technologies to further develop a net-centric architecture that is composable, modular, and Web-enabled. In other words, regardless of the source data obtained (historical, observed, real-time analysis) or how the sonar was modeled (sonar model, platform, local/remote), the resultant data must be intuitively visualized in a standardized and Web-capable manner.

It was determined early on in this research that five steps would need to be taken to accomplish this goal. First, some sort of decision aid or engine would have to exist to navigate through the sonar modeling process. This engine could be accessed through a web page or an installed application. Second, a sonar model would have to be exposed via the web. Recursive Ray Acoustics (RRA) is an example of a Java-based open source software package that can be easily converted into a Web service, but others will have to be made available in order to truly benefit from a net-centric architecture. Third, the sonar prediction models will need environmental inputs to ingest. Fourth, the model output will have to be visualized in a standardized way independent of which model was used. In addition, these visualizations must be 3D, Web-enabled, and outwardly intuitive to the user. Finally, an architecture would have to be designed that can govern the integration of these sub-components. All five steps were pursued successfully and are described in detail in this thesis.

E.   ORGANIZATION

This thesis provides the information necessary to understand the need for a sonar modeling and simulation in the tactical world, the existing technologies that can bring it into realization, the exemplar implementations of the architecture in Java, and the resulting Web-capable 3D visualizations. Chapter II, Background and Problem Statement, describes previous and current work in using Web technologies to bring modeling and simulation capabilities into the tactical realm. Chapter III, Related Work, discusses the challenges laid out by the CNO to foster interoperability throughout the

U.S. Navy, how net-centric sonar modeling fits into this challenge, and the goals that motivate this thesis work. Chapter IV, Net-centric Sonar Visualization reviews the overall net-centric architecture proposed in this thesis to create a seamless sonar modeling and simulation framework. Chapter V, Leveraged Web Technologies, walks the reader through the technologies used in this thesis to implement this architecture. Chapter VI, Metcast Web Service, describes a Web service solution to unlocking disparate sonar modeling assets. Chapter VII, Example Implementation, provides a look at an exemplar application that demonstrates how this architecture can be utilized to connect sonar modeling resources. Chapter VIII, Sonar Models, dissects the sonar models used in this thesis. Chapter IX, Sonar Visualization Design, explores the new and innovative visualizations designed in this thesis. Chapter X, Conclusions and Recommendations, summarizes the thesis contributions and provides information about improvements and extensions that can be made to the architecture and implementations. Appendices A through I provide examples of the more vital code elements in the Metcast Web Service, Sonar Visualization Panel, and Sonar Visualization API. Appendix J is the Sonar Modeling Schema used for the initial implementation of the architecture as defined by this thesis.

THIS PAGE INTENTIONALLY LEFT BLANK

II. BACKGROUND AND PROBLEM STATEMENT

A.   INTRODUCTION

Understanding the background and academic environment during the initial stages of this research helps convey the fundamental motivation for its genesis as well as its relevance to other projects and the U.S. Navy as a whole. Specifically, this chapter discusses the challenges laid out by the CNO to foster interoperability throughout the U.S. Navy, how net-centric modeling fits into the challenge, and incorporates these questions into the problem statement of this research.

B.   OVERVIEW

The forces that have shaped the research conducted in this thesis have evolved from two separate but related problems that face the U.S. military as it prepares itself for tomorrow's fight. The first is rooted in the DoD-wide effort to capitalize on vast improvements in force connectivity to create a net-centric force that can communicate tactical information back and forth between systems, nodes and units. Armed with a significant information advantage, decision-makers on the frontlines can perform well inside the enemy's OODA (Observe, Orient, Decide, Act) Loop, thus gaining the tactical advantage. While current and future data systems are being streamlined to interoperate, the same technologies can be used to provide new capabilities to the warfighter. Modeling and simulation assets, despite having significant operational value as a training and analysis tool, have been generally undeployable as a resource. Web-enabled modeling and simulation tools combined with the increased network connectivity, security, and bandwidth are changing this.

The second motivation is that, despite the existence of several excellent undersea warfare analysis, training, and decision tools, few have been able to find their way into operational use in actual battlespace. In a battle problem as complex as undersea warfare, it would be a great tactical advantage to conduct real-time analysis with real-time environmental data to conduct asset management, assess vulnerabilities and generate search/attack plans. Due to computing power limitations, it has been impractical thus far

to universally deploy such a system in the surface fleet. Programs such as the Interactive Multi-sensor Analysis Training – Personal Computer (PC-IMAT) are currently working to infuse real-time environmental prediction data into onboard decision tools (http://www.onr.navy.mil). As a complementary resource, it is beneficial to take the hardware and software requirements off the table and enable Web-enabled queries, analysis, and visualizations to be conducted remotely to land-based tactical assets over the Web.

C. FORCENET

1. Information Transformation

ForceNet is defined by CNO Admiral Vern Clark in *Sea Power 21*, his vision for Navy transformation, as the operational construct and architectural framework for naval warfare in the information age (Clark, 2002). ForceNet will integrate all stages of information processing. In an era where information is power, ForceNet will ensure the fleet has the integrated knowledge needed to dominate the battlespace. ForceNet will accomplish this through a net-centric architecture that connects the warrior to sensors, networks, weapons and decision tools. Standard joint protocols, common data packaging, seamless operability, and improved security are what will make implementing such a network possible. ForceNet is the "glue" that binds together the fundamental concepts of Sea Power 21: Sea Strike, Sea Shield, and Sea Basing.

To achieve and maintain warfighting superiority, the Navy must more effectively acquire, share and exploit information (Mayo and Nathman, 2003). Once information is acquired, it must be processed and shared with other units and systems to create a force-level operational advantage. Improved data-sharing systems, such as the Joint Tactical Information Distribution System (JTIDS), are vital to the sharing of tactical information in a net-centric force. An overabundance of information, however, does not prove to be advantageous if it is not properly exploited. A critical challenge for ForceNet is the development and integration of powerful decision aids that can swiftly translate into coordinated action. Figure 1 below is an excerpt from "ForceNet: Turning Information into Power," an article written by Vice Admiral (VADM) Richard Mayo and VADM John Nathman explaining the role of decision and analysis tools in ForceNet.

8

> "The goal of ForceNet is to arm our forces with superior knowledge, leading to increased combat power. In pursuit of this goal, ForceNet will provide a comprehensive network of sensors, analysis tools, and decision aids to support the full array of naval activities, from combat operations to logistics and personnel development. The focused, timely, and accurate data delivered by ForceNet will help leaders at every level by allowing them to draw on vast amounts of information and share the resultant understanding. This will increase the joint force's ability to synchronize activities throughout the battle space to achieve the greatest impact."

Figure 1. Excerpt from *Proceedings* article by VADM R. Mayo and VADM J. Nathman on the role of decision and analysis tools in the ForceNet architecture (Mayo and Nathman, 2003).

In the world of USW, programs are beginning to emerge that are attempting to realize these goals. However, due to the complex nature of USW, this is not a trivial task. The underwater environment is complex, quite variable, and absolutely necessary to grasp in order to properly plan and conduct effective undersea operations. Common USW planning tasks such as buoy field operations, asset allocation, and vulnerability assessment can not be accurately done without understanding how sonar will perform in a specific environment. To achieve this understanding requires running environment-specific oceanographic data through high-performance sonar prediction models. In the past, this required prior knowledge of an operational area and land-based supercomputing power. More current systems have relied on moderately powerful but deployable computing assets and historical data to generate sonar predictions. While these have been and continue to be valuable resources, one can't help but feel that much potential is going unrealized due to the lack of integration between land-to-shore, ship-to-aircraft, and analytical-to-tactical systems.

ForceNet is the solution. USW decision aids are currently under development to integrate sensors, modeling and simulation assets, decision aids, and weapons systems. These tools will help bring net-centric warfare to fruition. It is also important, however, to realize that no system is perfect. Another great advantage of ForceNet is choice. Imagine a scenario where a warfighter is prosecuting a hostile submarine using an

onboard USW TDA updated with real-time sensor data while a sonar supervisor is simulating the effectiveness of potential buoy patterns by remotely accessing land-based supercomputer grids over the Web. This sort of complete system interoperability will allow warfighter teams to carry out missions using a variety of available tools. This is ForceNet.

2.   Open Standards, Open Source

The Department of Defense (DoD) recognizes the power of the commercial Internet Protocol (IP)-based informational environment. Advances in computing speeds, Web technologies, and remote computing infrastructure have transformed the Web into an important element of the Global Information Grid (GIG) and a key enabler of net-centric warfare. The realization of a dominant, net-centric force is dependent upon its ability to exploit the GIG. The commercial sector has taken command of the information technology (IT) world. The gap between the DoD acquisition influence and the global IT market has made all of the tactical data and software based systems of U.S. Navy ships and aircraft unaffordable to maintain and update (Rushton and others, 2004). The key to breaking out of this cycle is to adopt and enforce an open architecture. An open architecture is one which is governed by public specifications that either can be officially approved or privately designed. The idea behind open architecture is that, since the specifications are public, anyone can design add-on products to any standards-compliant application. Possibly more importantly, future applications can be designed to freely interoperate with that system by conforming to that standard.

As future fleet programs migrate to an open architecture, the current fleet is left to cope with isolated legacy systems that do not fit into the ForceNet puzzle. This may not necessarily be the case. Current Web technologies have the capability to replace middleware. Middleware is software that allows otherwise disparate systems to communicate to each other. Extensible Markup Language (XML) is an open standard that is a well-proven medium for communicating across languages and platforms. As you will see demonstrated in this thesis, a simple Internet Explorer browser on a Windows PC has the ability to communicate and run high-resolution sonar modeling algorithms on a Linux supercomputer and be able to understand and display the results. By using open

10

standards as a common language, legacy systems can be transformed into interoperable pieces of a net-centric architecture. If ForceNet is the "glue" that will hold it all together, open architecture is the "recipe" that's used to make the glue.

In addition to complying with open standards, all code developed in this thesis is open source. Unless it contains sensitive information, all code developed by the MOVES Institute follows an open source policy. This implies all source code is made available to the general public for use or modification from its original design free of charge. An open source policy encourages collaboration and provides relief to institutions dependent on proprietary software. By developing in open source, the customer is spared being locked-in to proprietary dependencies that only ensure future expenses, limited extensibility, and no data control. If the source code is sensitive or classified, open-source software can be offered as restricted distributions. Most classified data is time-sensitive, location-dependant, or platform specific. Hence, classified parameters can be kept in XML configuration files, leaving code bases unclassified. The WEAVER site, developed in partnership with Sonalysts as part of this thesis, was developed as an unclassified resource using low-resolution, arbitrary datasets. The software package was then seamlessly transferred to a classified network and began ingesting "live" data.

D.   A MISSING PIECE
1.   Infusing Modeling and Simulation into the Tactical World

By declaring ForceNet as the paradigm that will guide the Navy's transformation into the information age, a door is opened that allows resources previously limited to shore commands or possibly aircraft carriers to become a part of the deployed warrior's tactical toolkit. Modeling and simulation certainly has its place in the DoD. It has helped train countless fighter pilots, firefighters, and shiphandlers. Commanders have tested out new tactical maneuvers and wargames have brought previously unthought-of vulnerabilities to the forefront. Advances in computing technology have allowed simulation assets such as onboard trainers to deploy with fleet assets. However, the modeling world has always had the stigma of being "practice," something done to prepare, to train, and to experiment. The closest modeling and simulation has come to the battlespace is the forecasting capabilities that weather or sonar modeling resources

generate that may have an affect on the tactical battleplan. The computing power necessary to conduct these large-scale modeling tasks is not practical on most naval platforms. A sudden change in anything may relegate all the prior modeling irrelevant.

This historic disconnect between tactical and simulation systems is quickly changing. XML-based languages, Internet technologies, and Web services are enabling a new generation of distributed modeling and simulation applications to emerge, develop and interoperate (Brutzman and others, 2002). Embracing Web technologies as a shared-communications platform and a reliance on well-supported open standards can bring high-end modeling and simulation right into the thick of battle. In the near future, a Platoon Commander will receive a weather alert on his Personal Data Assistant (PDA) that a sandstorm will be inbound to his position in twenty minutes. A staff officer in theater will run a planned mission over and over to identify key vulnerabilities or potential flaws in his preparations. In work related to this thesis, a frigate sailor operating independently in the vicinity of a possible hostile submarine will remotely access oceanographic analysis data and run high-resolution sonar prediction models from a Web browser to optimally lay out a buoy pattern. ForceNet will be realized when distributed sensors; land-based modeling, simulation and analysis tools; onboard decision aids and deployed weapons systems are all incorporated into the same detect-to-engage sequence.

2. Sonar Modeling and Visualization

One of the mission areas that has the most to gain through the integration of modeling and simulation into the tactical picture is Undersea Warfare (USW). The underwater picture is extremely complex and its mastery is crucial to effective operations. The performance of sonar is highly variable and greatly dependent on the given environmental conditions. A highly effective plan today in one operational area may be worthless in twelve hours or just a few miles away. In addition, a multitude of factors and their interactions play a role in determining sonar performance. Sea temperature, bottom depth, bottom type, salinity, surface conditions, and bioluminescence are a few that can have an affect. Dozens of sonar models with distinct and complicated algorithms are available and have value as predictive tools, but few answer every bell (Etter, 2004). Already, one can see that sonar modeling is a

complicated business any time, let alone with a submarine closing in for the kill. The good news is all these tasks, including manipulating high-order mathematical equations and sifting through model decision matrices, are what computers do best. The trick is, conveying what the computers already calculate into accessible products that a warrior can understand and use.

Sonar visualization is not a perfected science. Visualizing anything in three dimensions (3D) has never been easy. First, typically in the past 3D objects had to be pared down to two-dimensional (2D) viewers. Cross-sections of data can give a snap-shot of information, but do not provide the entire picture. Second, visualizing a 3D object such as an apple does not at first appear to be a difficult task. However, what if the user needs to know if there is a worm on the inside or where all the bruises are, this becomes a much more challenging endeavor. This problem relates directly to sonar visualization. As the sonar pulse moves through 3D space, its strength (and thus, ability to reflect off a submarine and travel to a receiver) is diminished. It is tactically important to know where in 3D space dead-spots, or shadow zones, are located. This is an interesting visualization problem because in 3D, data tends to block out other data. A third difficulty is that a sonar pulse is a kinetic event. A static representation, or "cloud" of data, may be useful in determining coverage areas, but does not necessarily convey to the operator how the sonar is reacting to the environment. Only a dynamic visualization that shows the pulse as it propagates and interacts with the environment can intuitively display this information. Operator exploration of this data space using 2D/3D scientific visualization techniques is expected to be a critically valuable new capability.

E.   SUMMARY

The desire for a deployable undersea warfare (USW) modeling and simulation toolkit and the implications of U.S. Navy's transformation into a net-centric force call for the development of an architecture aimed at fostering net-centric USW decision aids. Through ForceNet, the U.S. Navy intends on leveraging current Web technologies to bring unprecedented information-processing support to the warfighter. By adhering to these principles, deployable USW decision aids can be designed to take advantage of high-resolution shore-based resources through the power of Web Services. As a result,

13

deployed USW systems can reach back to powerful supercomputing grids to conduct real-time, high-resolution operational analysis.

III. RELATED WORK

A. INTRODUCTION

This chapter discusses other work going on the field of sonar visualization, Web-based modeling and simulation, and the design of net-centric architectures. The incorporation of modeling and analysis resources into the tactical information loop is currently being pursued at NPS and elsewhere. From innovative applications to compression algorithms to governing frameworks, many moving parts are coming together nicely to demonstrate the true power of the interoperability inherent in XML and open standards.

B. OVERVIEW

The individual technologies exhibited in this thesis are not pioneering. The concept of Web services and net-centric interoperability is a well-proven and highly-successful model in the corporate world. What is innovative, however, is the combination and transfer of these technologies into the tactical realm to provide deployed warfighters the ability to operate in an information-on-demand environment.

One area of this net-centric transformation where great progress is being made is in integrating modeling and simulation technologies into the tactical environment. This chapter summarizes some of the work that is being done in this area in order to provide the motivation and background needed by the reader. The overarching principles that guide the main effort of this thesis are outlined in the Extensible Modeling and Simulation Framework (XMSF). The XMSF group is working towards leveraging existing web technology to make web-based modeling and simulation accessible from the front lines.

The Web-Enabled Architecture for Visualization, Evaluation and Research (WEAVER) is a Naval Air Warfare Center (NAVAIR) sponsored project that provides the primary motivation for this thesis. WEAVER allows an operator on a deployed or ashore unit to use a SIPRNET web browser to retrieve real-time environmental data from massive databases, remotely run sonar modeling and prediction algorithms on land-based

supercomputers, and view autogenerated interactive 3D displays in order to support a number of underwater visualization operations. The technologies researched in this thesis that were incorporated into WEAVER were first implemented in the Autonomous Underwater Vehicle (AUV) Workbench, a mission planning tool for autonomous underwater vehicles. The AUV Workbench offers much of the same capabilities as WEAVER (as a small fraction of its overall functionality) only via an imbedded Java Swing graphical user interface (GUI).

NPS is not the sole investigator of sonar visualization techniques, web-based modeling and simulation, or the tactical application of web services. However, it is one of the few attempting to find synergistic value in merging all three. This chapter also takes a look at recent research going on in the field of sonar visualization and techniques used in the prevalent TDAs in fleet use. Finally, a brief synopsis is given outlining how the US Army is approaching similar solutions. The Army Research Laboratory is researching the use of existing web-service technologies to provide weather data on demand over the web directly and as input into larger tactical systems.

C.   EXTENSIBLE MODELING AND SIMULATION FRAMEWORK (XMSF)

The XMSF project is focused on leveraging existing web technologies as a shared-communications platform and information delivery framework. In short, XMSF is composed of a set of standards designed to govern web-based modeling and simulation. In the world of XMSF, XML is a sine qua non of Web-based simulation. That is, it defines the structure of how information is represented, validates that the information to be passed conforms to that structure, and itself comprises the message to be transferred. The self-describing nature of XML allows it to bring legacy systems into the realm of net-centric warfare and the interoperability it provides.

XMSF has five major precepts as outlined by (Brutzman and others, 2002). The first is that web-based technologies can enable a new generation of modeling and simulation applications to emerge, develop and interoperate. Any new applications conforming to XML-based frameworks such as XMSF are likely to achieve

unprecedented levels of interoperability. New applications are expected to achieve great synergies through the capability of freely passing data back and forth between systems.

Support for operational tactical systems such as decision aids from modeling and simulation (M&S) is essential but is currently almost nonexistent. The adoption of a common framework such as XMSF could greatly increase the capability and accuracy of forward deployed TDAs. One of the goals of this thesis and the Sonar Visualization Project is to demonstrate the potential of allowing client-side applications such as the AUV Workbench to reach back to high-performance supercomputing grids, either to access real-time weather and oceanographic data for analysis or run high-resolution sonar modeling algorithms to get up-to-the-minute sonar prediction data.

An extensible framework for XML-based languages can serve as a bridge between forthcoming modeling systems and open/commercial web standards as well as continuing to support legacy systems. The great advantage of XML is that it is platform and language independent. This means developers and users can connect previously disparate systems into one interoperable network. Not only will new systems be able to communicate with each other over the network, but older "stovepipe" applications (i.e. systems unable to communicate with more than one other system) would also be included. As demonstrated in this thesis, web services can be designed as "translators" that accept and return XML to the client, but translate the XML into the host systems native language. Wrapping a translator as the external interface allows a legacy system to communicate via XML without having to change outdated or proprietary code.

A fourth precept of XMSF is that compatible and complimentary technical approaches are now possible for model definition, simulation execution, networked-based education, network scalability, and 2D or 3D graphics views. This is where the power of synergy comes into play. By sharing one language for every requirement, each entity can multiply its effectiveness by tapping into other systems. In this thesis, a framework is created to define oceanographic data that is used as inputs into sonar modeling algorithms. Also in this thesis, that same framework is used by a sonar visualization module that is then able to display this data in 3D worlds. This can be extended to any system that might benefit from having oceanographic data. Sonar operator trainers,

17

oceanographers, even environmental groups might design applications that can now access and understand this data. This free flow of information is the crux of the net-centric argument.

Finally, following a web approach for future technology and content production provides best business cases from a world-wide perspective. These technologies have been embraced by the public sector and are well-supported. With these precepts kept close at hand, the XMSF group believes that this framework has the potential to support the needs of the modeling and simulation community and evolve into DoD-wide real-time tactical support resources.

D. NPS AUTONOMOUS UNDERWATER VEHICLE (AUV) WORKBENCH

The AUV Workbench is an open-source XMSF project under continual development by NPS. As vehicles, procedures, and tactics continue to become more refined, it has become apparent that there is currently a lack of dedicated tools and data archives to support collaborative AUV research. This lack of interoperability and system integration continues to impede field progress and development of universal planning tools. The development of an Autonomous Vehicle Control Language (AVCL) can provide a common mission and data formatting language using XML that will facilitate AUV integration and mission planning tools.

One of these mission planning tools is the AUV Workbench. Since AUVs are relatively expensive to build and operate in perilous environments, they are ideal use cases for simulators that can accurately and repeatedly test missions in a virtual environment. Due to these factors, the AUV Workbench can reap significant value as a mission planning and rehearsal tool. By allowing an operator to check for errors and mission coherence before putting the AUV in the water, the AUV Workbench can perform a "sanity check" on an AUV's flight-path that may prevent a previously unforeseen catastrophic incident. While creating an AUV planning tool is certainly not a new idea, it would be beneficial to have a tool that can integrate multiple and disparate AUVs. Tactically deploying multiple AUVs is a complex operation due to the lack of interoperability between disparate AUVs and universal planning tools. In addition, it is

inefficient and costly for larger organizations to maintain multiple planning tools that only work for each system's particular vehicle.

As described in (Brutzman and others, 2003), the AUV Workbench provides a mission planning, rehearsal and replay capability that can solve some of these issues. It is open source, written in Java and available for download on the web via either an executable autoinstaller or a Concurrent Versioning System (CVS) source code base. An operator can create, modify, or archive XML-based mission scripts through a Java Swing graphical user interface (GUI). The mission can then be viewed in an embedded Xj3D browser. Xj3D is an open source toolkit that allows a user to create a browser or window capable of viewing files that comply with the Virtual Reality Modeling Language (VRML97) or X3D specifications. These files describe a three dimensional (3D) world that can be freely navigated by the user. The real value of the AUV Workbench is that the mission execution thread runs the software from the actual AUV. Thus, shortfalls in the mission script can be immediately identified during testing prior to putting a vehicle in the water. It also contains a virtual world dynamics thread that implements a physics-based hydrodynamic mathematical model.

The Xj3D browser is used to display the 3D mission scenes. Operators can interactively navigate the scene as the AUV conducts its mission, using a mouse or by visiting predetermined viewpoints. Since the scene is capable of displaying bathymetry data and multiple AUVs running simultaneously, an operator can determine if an AUV's mission script places it in any danger of collision or grounding. Figure 2 shows a typical AUV Workbench display.
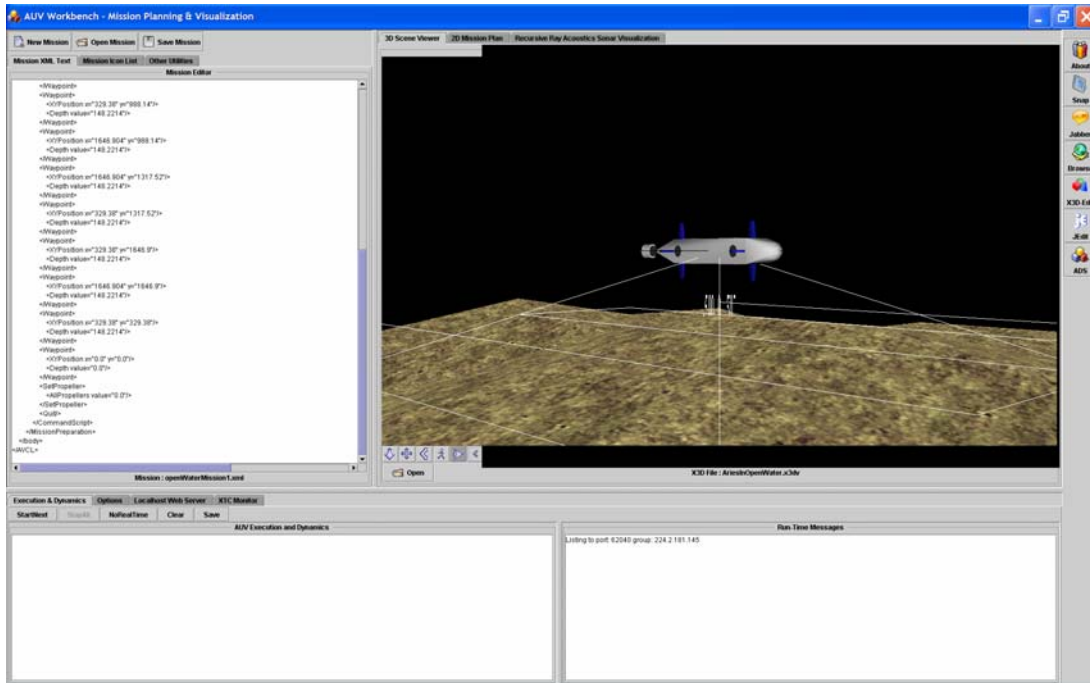
Figure 2. The AUV Workbench allows a user to view an AUV mission in 3D. Windows, clockwise from upper left: Mission Script, 3D View, Real-Time Messages, and Robot Execution/Dynamics Output.

The latest addition to the AUV Workbench is a sonar visualization panel that actualizes the research completed in this thesis. The Sonar Visualization Tab brings up an embedded Xj3D browser with its own GUI panel. Directly from the user panel, the user can retrieve oceanographic data from a web service that translates the binary data into XML. This allows the user to view 3D bathymetry data and environmental data plots. The user can then describe a sonar source using GUI inputs or selecting an archived XML file that can be used in conjunction with the oceanographic data to run local or remote sonar prediction models and display auto-generated 3D scenes. Future work includes incorporating real-time sonar prediction data into the AUV mission scenes to predict sonar coverage areas and probabilities of detection.

20
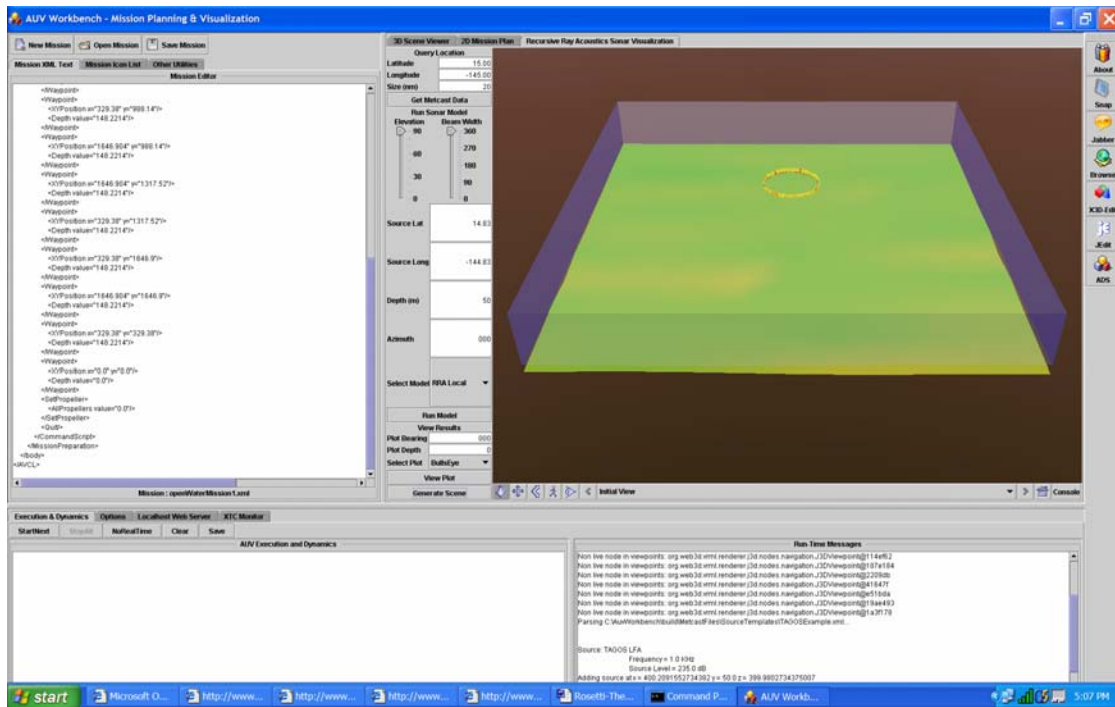
Figure 3.   The Sonar Visualization tab of the AUV Workbench allows a user to model and visualize ocean bathymetry and sonar performance in 3D.

E.   WEB-ENABLED ARCHITECTURE FOR VISUALIZATION EVALUATION AND RESEARCH (WEAVER) PROJECT

NAVAIR, Sonalysts Inc., the NPS MOVES Institute, and Fleet Numerical Meteorology and Oceanography Command (FnMOC) have collaborated on a project to develop an undersea warfare (USW) modeling and simulation TDA. This decision aid, appropriately named WEAVER, is a fully net-centric and multi-platform system through its implementation using XMSF technologies. The unique concept behind the development of WEAVER is to create a Web-based focal point that can provide access to high-performance sonar modeling resources via a simple web browser. By using XML as an intermediary, each stage in the sonar prediction process is modularized. That is, the system can access any number of previously disparate resources and still produce standardized output for the user regardless of the model used.

While harmonizing interfaces is a promising venture in itself, a further innovation is that all these resources are orchestrated through a web browser. Not only are WEAVER requests initialized from the site and made to remote computing stations via remote Web Services procedure calls, but the resultant output is also made available on

21

the web page via a hyperlink. Since no client is needed, no software or hardware support will be required out in the fleet. The only requirement to access the tactical decision aid is a SIPRNET connection and a browser plug-in that can be downloaded from the network.

The user may choose which sonar model to use or the model decision engine will heuristically select the sonar model and platform automatically. After a series of inputs is entered by the user via a web form, the server will initiate an XML remote procedure call to the Metcast Web Service (MWS) developed in this thesis project. The MWS will provide an XML response consisting of relevant oceanographic data that will be automatically ingested by the selected sonar prediction model. The model output will be converted into schema-governed XML. The response is then parsed and used to create simple 2D and complex 3D virtual worlds that can be interactively navigated by the user.

While the WEAVER project was the initial motivation for much of the thesis research, most of the development occurred in the AUV Workbench. After the technology was proven successful in the AUV Workbench, the MWS and Sonar Visualization Package were installed and integrated with the WEAVER web server.

F.   XML SCHEMA-BASED BINARY COMPRESSION (XSBC)

XML, by nature of its original design criteria, is verbose. Since XML uses tags to delimit the data, XML files are almost always larger than their binary counterparts. Translating a binary file into text-based XML can be cumbersome and in some cases doubling or even tripling in size. Although bandwidth availability is growing on military networks, it can never seem to outpace demand. Thus, it is desirable to minimize bandwidth requirements needed for passing data over a network. This is even more paramount for AUV communication that may rely on underwater acoustic modems that operate in noisy environments. This has brought about concerns over the usability of XML due to its greater size. XML Schema-Based Compression (XSBC) can not only counter the verboseness of XML, but actually can generate compressed XML files that are smaller than the original non-XML files in a compressed, self-validating binary format.

22

Although XML is relatively verbose, it is also rigidly structured and standardized. This characteristic of XML translates well into compression algorithms. XSBC takes advantage of this feature and offers a general approach to the binary serialization of XML documents. In XSBC, XML elements and attributes are replaced via a tokenization scheme while the inherent data is preserved. After XSBC is applied, the data file that had ballooned in size after XML conversion typically decreases in size to roughly three-quarters the size of the original binary version. The added structure of XML lets custom compression algorithms work more efficiently than general compressors such as zip or gzip. These savings can be applied directly or can be used to add another level of data assurance such as Forward Error Correction (FEC) using Hamming codes.
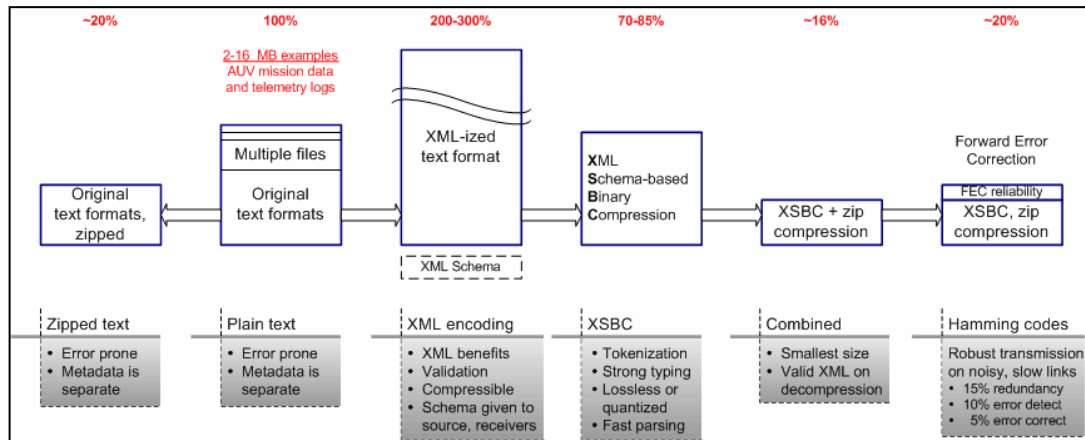
 Figure 4.   XSBC results as compared to standard compression algorithms.

G.   SONAR MODELING AND VISUALIZATION

Sonar modeling attempts to recreate the highly variable ocean acoustic environment and predict how a sonar pulse may perform. This information can be invaluable to undersea mission planners. However, many distinct sonar models exist due to the complexity of acoustic propagation prediction in the undersea environment. Unfortunately, mission planners rarely find themselves in a position to pick and choose which sonar model to use for a particular scenario. Several TDAs relevant to USW have been developed to take much of this guesswork out of sonar prediction. These TDAs must be able to effectively convey resultant data to the user in an intuitive way. Data

visualization has been an active research topic for many years and continues to be a focus in undersea warfare application development. This section identifies related work in sonar modeling and visualization.

1.   Interactive Multisensor Analysis Training (IMAT)

The IMAT project's mission is to integrate training, mission rehearsal, tactical execution, and post-mission analysis tools to develop and maintain mission critical skills throughout the fleet. IMAT was originally developed under sponsorship of the Office of Naval Research (ONR) to support individual training in classroom environments. IMAT itself is a concept-based instructional tool that is designed to promote context-based learning in an intuitive and familiar multimedia environment. IMAT relies on high-resolution databases and scientific visualization to illustrate the interaction between acoustic variables. Several acoustic models are available to generate propagation loss results, including the Comprehensive Acoustic Simulation System (CASS), Parabolic Equation (PE), and Acoustic Transmission Loss (ASTRAL) models (http://www.onr.navy.mil).
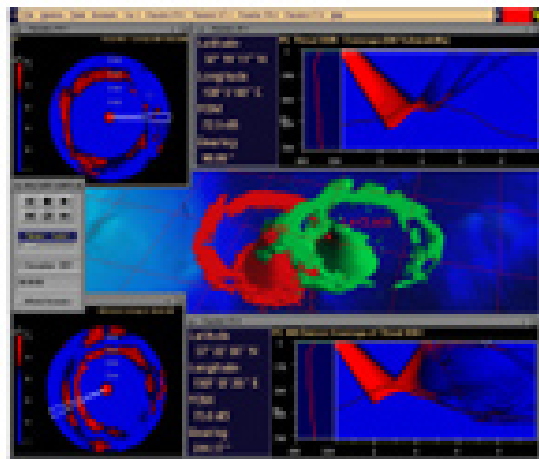
Figure 5.   The Integrated Multi-sensor Analysis Training (IMAT) system features high-resolution 3D visualizations to enhance user understanding (http://www.onr.navy.mil/sci_tech/personnel/342/training/majapps/imat).

PC-IMAT, a version of IMAT deployable on a laptop or desktop computer, is the standard TDA for acoustic sensor deployment and search planning on all US submarines.

PC-IMAT supports both structured lesson training and interactive environmental acoustic modeling. PC-IMAT is used at sea for acoustic modeling in tactical situations. Embedded models and databases allow the user to select geographic locations and times of year to predict sonar performance in a variety of tactical contexts.

The PC-IMAT Destroyer Squadron Anti-Submarine Warfare (ASW) Support System is being proposed as the standard tactical decision aid for the surface fleet. Taking advantage of a reliable SIPRNET, contact positioning and sensor data can be passed back to the PC-IMAT Analysis Display System. The shore-based application can then develop mission plans based on real-world environment and threat and coordinate sensor employment and operational tasking. This capability allows decision-makers to sustain situation awareness throughout the mission and adapt to changing environmental conditions.
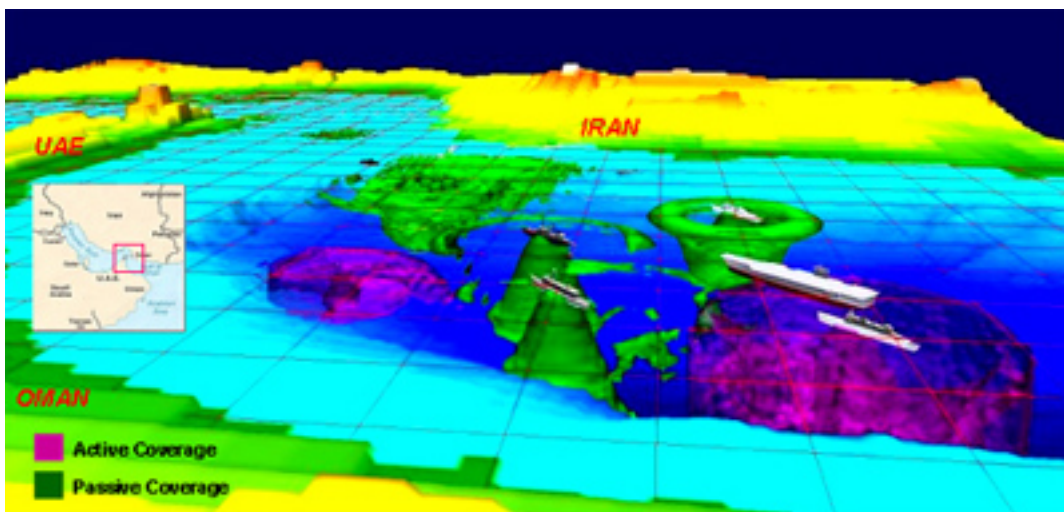
 Figure 6.   The Integrated Multi-sensor Analysis Training (IMAT) system uses overlays, 3D visualizations and historical data to create interactive training scenarios (http://www.onr.navy.mil/sci_tech/personnel/342/training/majapps/imat).

PC-IMAT developers continue to produce new and intuitive visualizations. These visualizations, although only viewable within an IMAT-compliant viewer, provide detailed representations of threat-sensor-environment relationships that are invoked as needed for both training and tactical decision aids. PC-IMAT visualizations include passive/active sensor performance prediction, coordinated operations, and multi-static

acoustics. Visualizations include single line of bearing plots, bullseye plots, and multi-static coverage plots. The basin-wide analysis display in PC-IMAT is an overview of sensor performance for a geographical region and is used to plan sensor allocation. This display includes passive sensor coverage and ownship vulnerability estimates for a variety of sensor configurations, geometries and threats.

2.   USW Decision Support System (DSS)

Formerly known as the Common Undersea Picture (CUP), the USW DSS program is devoted to the development of a single suite of decision support and collaborative tools that will provide a shared situational awareness of the underwater battlespace. Fielding USW DSS is expected to provide fleet operators with the tools and capabilities to properly align sensors to exploit the underwater environment, tactically manage waterspace, efficiently allocate resources, and assess vulnerabilities in order to better plan and conduct USW operations. The "common picture" includes shared environmental data, common decision aids and acoustic models, and standard communication channels and databases services. By adhering to a net-centric architecture, DSS will be a cross-platform system (http://www.dtic.mil/ndia/2001interop/thompson.pdf).

DSS developers have outlined a four-step process to achieve network-centric capability. The first step is to establish connectivity to tactical decision aids that are already available. Then, these applications will be modified to ingest in situ data as apposed to historical or modeled data. The third step is to establish methods for extraction and ingestion of data from platform tactical systems. This will require access to tactical systems on all undersea warfare capable platforms and all data types. Finally, an automated cross-platform data exchange system will be developed to manage information flow with regard to security, compression, filtering and prioritization (http://www.dtic.mil/ndia/2001interop/thompson.pdf). The DSS program, if implemented correctly, will solve many of the current shortcomings with current USW decision aids by consolidating disparate system, infusing real-time data and integrating all undersea platforms.

H. COMMAND AND CONTROL INFORMATION EXCHANGE DATA MODEL (C2IEDM)

The Command and Control Information Exchange Data Model (C2IEDM) is a common interface for the exchange of battlespace information (Hodges, 2004). Through a generic representation of command and control information, C2IEDM can support the transfer of vital information between services and multinational forces. Current work by (Hodges, 2004) provides an exemplar demonstrating successful transformation of unit data from the C2IEDM format to one accepted by a simulation package using the Extensible Stylesheet for Transformation (XSLT), thus fostering interoperability between simulation tools and Command and Control (C2) systems. In addition, a comparison is made between a document-centric version of the C2IEDM that uses XML to represent unit data as opposed to a database-driven version. (Hodges, 2004) concluded that an XML-based representation facilitated an easier manipulation mechanism to integrate simulation packages with C2 systems.

I. NET-CENTRIC MOBILE WEATHER TECHNOLOGY

The Sonar Visualization Project at NPS is not the only Department of Defense research aimed at providing tactically relevant data to the warfighter on demand via a net-centric architecture. The Army Research Laboratory's Battlefield Environment Division is currently working towards leveraging current advances in information technology to make weather intelligence products (such as decision aids, alerts, and map overlays) available to the lowest echelons. Weather possesses the ability to have a profound impact on military operations and has the capability to change very rapidly. Thus, there is a great need to get as much weather data to the deployed warfighter in real-time. This is possible using a net-centric approach.

As an Army research laboratory, its focus is on putting weather information into the hands of the soldiers on the ground. Relying on the cross-platform capability of Java, several weather applications have successfully been tested on Toshiba Personal Data Assistants (PDAs). The following is a brief synopsis of some of the mobile weather technologies the group has been able to successfully demonstrate as summarized in (Sauter and others, 2003). The Integrated Weather Effects Decision Aid (IWEDA)

provides graphical and text information regarding weather impacts on hundreds of friendly and threat weapon systems via a web browser. The Weather Alert Subscription application allows a mobile user to receive weather related alerts via text message using Java-based remote method invocation (RMI) servers. The Local Observation Broadcast (LOB) system provides the capability to enter and send local weather observation data back to a remote server implemented as a web service. The LOB data can then be used as initialization data for a prognostic meteorological model or as an input to a Chemical, Biological, Radiological (CBR) hazard dispersion model. Mobile Acoustic Detection allows a user to make Java-based remote procedure calls (RPCs) to a server that determines probability of detection based on a specified acoustic target and sensor. These projects outline some of the initial steps the Army is taking to create a net-centric flow of information that ties in the soldier on the ground to nearly limitless computation support and integration. These approached are complementary to and compatible with Navy ForceNet approaches.

J.   XML TACTICAL CHAT (XTC)

XML Tactical Chat (XTC) is an open-source technology being investigated by NPS under XMSF that can potentially completely upgrade and restructure all tactical military communications. Many chat technologies used today, such as IRC, Yahoo, and MSN, are proprietary and do not fulfill desired interoperability requirements. These programs are often blocked by firewalls, lack robust interfaces, are unsecure, have costly licenses and/or are not governed by open standards. Instead of an arbitrary selection of proprietary chat applications that only partly fulfill desired requirements, a better solution might be to adopt an XML-based chat protocol that robustly defines interfaces, data structures, and administrative parameters. XTC is an open standards solution to this problem. It uses Jabber, an open-source chat protocol, to drive the chat engine. XTC is XML-based, thus it is Web-ready, fully interoperable, firewall friendly, and secure. In addition, XSLT can be used to translate between languages, systems, and applications. For example, a chat-enabled TDA may automatically send a chat message to a USW chat group alerting all members that a mine was detected. This sort of functionality can firmly chat into the tactical picture.

28

K.   SUMMARY

Much work is currently being done to advance the fields of sonar modeling and Web-based simulation. XMSF governs Web-based modeling and simulation through the use of standards such as XML. The AUV Workbench is a mission planning tool designed within XMSF to allow for the networked simulation of AUV missions. XSBC allows for the efficient transportation of XML over bandwidth-scarce networks. IMAT and DSS are existing sonar modeling assets that are continually being improved to meet the requirements of the information age. Other services such as the U.S. Army are researching similar technologies to improve tactical decisions on the battlefield by providing the warfighter with real-time weather and intelligence information.

THIS PAGE INTENTIONALLY LEFT BLANK

IV. NET-CENTRIC SONAR VISUALIZATION

A. INTRODUCTION

This chapter delineates a net-centric architecture developed as part of this thesis to govern how sonar modeling tasks might be distributed and define the interfaces to connect them. The difficulty in designing a deployable USW decision tool lies in the fact that the best resources to effectively perform these tasks (including the decision-maker) are not collocated. By defining a net-centric sonar modeling and simulation architecture, these resources can be tied together and made available to the warfighter. In addition, by creating and publishing an open architecture, future and current sonar modeling assets can seamlessly interoperate as one system.

B. NET-CENTRIC SONAR MODELING ARCHITECTURE

By adapting a net-centric architecture, several advantages can be gained in the field of sonar modeling and in USW in general. First and foremost, the development and adoption of a common language for USW can provide a dramatic force multiplier. Previously disparate sensors, command and control systems, decision aids, and weapon systems will be able to freely pass and understand data using schema-governed XML as the universal language. This paradigm can be specifically applied to sonar modeling and visualization. A schema describing sonar modeling data, as a subset to a greater undersea warfare schema, has the potential to greatly increase the interoperability of sonar prediction resources with each other and with tactical systems to create a common operating picture.

In one potential use-case, a client (which can just be an internet browser) is able to access a Web service to pull oceanographic data (or push observed data) in XML form and validate it against a published schema. This information can then be prepared for display or used as initialization data for a sonar prediction model. Because the XML conforms to known schema, the client would is thus able to choose which model to utilize depending on situation, availability or connectivity. Tools can then use a local decision aid such as PC-IMAT or one of many sonar prediction models that can installed on supercomputer grids exposed on the Web. In the case of a remote model, the Web

service can return an XML stream that again conforms to the undersea warfare schema. Because it conforms to a standard, a single viewer can visualize this data in a consistent manner regardless of which model was used. Thus, the client understands and creates the same plots whether CASS/GRAB or RRA was used as a model.

The common objection to such a proposed architecture is that these legacy sonar prediction models and decision aids do not all understand XML and therefore need to be rewritten. This is not the case. A relatively straightforward solution is to develop a Web service interface for each model. These Web services act as translators by taking in XML inputs and converting them into the native language of the legacy system. Once the legacy system generates output, the Web service converts the results back into XML for return to the client. In this design, the wrapping XML-based web service is mostly (or completely) external and the legacy system is never exposed to XML, thus it does not require any code manipulation by any proprietary entities. This was the approach taken to expose the FnMOC Metcast server and the CASS/GRAB sonar model as proof-of-concept tasks for the WEAVER project. By creating Web service interfaces for these resources, clients can be developed with remote access to high-resolution oceanographic analysis and sonar prediction results. The client will be returned XML and can now visualize this data in a standardized way independent of which sonar model generated the data.

As stated previously, the goal of this thesis is not to replace existing decision aids. Several programs are in progress designed to integrate undersea warfare resources. Some of these ventures are using Web services. This is true for PC-IMAT, DSS, and the AUV Workbench. However, it will benefit the warfighter to possess alternate resources that can provide sonar modeling capabilities that do not require extensive software or hardware maintenance, upkeep, and storage. One useful realization of the net-centric architecture is to provide all these services through a standard web browser. An operator needing only a browser such as Internet Explorer (and currently a plug-in to view the X3D files) can manually input specific parameters such as location, size of the operating area, and type of sensor. The Web server then takes full responsibility for accessing the appropriate oceanographic data from a Web service, choosing which sonar prediction model to run based off a rule-based model decision engine, ensuring parameters are correctly setup,

32

and generating displays that the operator can click on and view right there in the browser. Generating the visualizations in X3D allows users to freely navigate though 3D data representations.

By exposing multiple sonar prediction models on the web and standardizing how each is accessed, an operator now has significantly more resources available to select from. Previously, a deployed operator was restricted to requesting prediction data well in advance of an operation, relying on shore-based support to feed model results out to the deployed platform, or conducting modeling onboard with sparse computational resources and far less data using a tool such as PC-IMAT. There are many sonar models in existence, and many are extremely accurate in some environments and erratic in others. The most common models in use today are those that are relatively coherent in most environments. Some models do not handle certain sonar sources well or not at all. It would be advantageous to be able to select the optimal model based on environment, operation, or availability of the model. As more models are exposed using XML interfaces, this becomes possible. Web technology can thus provide the operator with several options that were previously unavailable.

Although supercomputers are now reaching relatively manageable sizes, it is still inconvenient or impossible to outfit each ship and aircraft with its own high-performance supercomputer grid. A net-centric implementation allows thin clients and even regular Web browsers to become user interfaces for large-scale supercomputers. An operator at sea can run high-resolution sonar prediction and oceanographic models remotely from a personal computer on a ship or a laptop in an aircraft. The XML-RPC functionality that is inherent with SOAP allows for distributed computing to become commonplace. In this architecture, any application or web server has the ability to run high performance computing algorithms located on a large supercomputer grid located halfway around the world. Connecting supercomputers to the tactical decision-making loop is an important innovation.

The concept behind ForceNet is to develop an architecture that incorporates tactical sensors, weapon systems, decision aid and other combat entities into a comprehensive system that can seamlessly pass data back and forth. The data sharing

capability necessary for this integration is made possible by XML. This holds true for sonar visualization. Internally, this project incorporates previously disparate systems such as Metcast, CASS/GRAB, and sonar visualization servers by taking advantage of Web service technologies. Through the development of an overarching undersea warfare schema, which the research in this thesis consists of only a small part, the resources in this sonar architecture can be extended into other decision aids as well. For example, signal excess zones for a particular sonar sensor may be displayed in a Tactical Action Officer (TAO) TDA.

Adopting a Web-based architecture is not yet a complete solution. Although it carries with it massive potential to revolutionize how the fleet conducts business, there are still some issues that need to be addressed. The first of these is a concern about bandwidth availability. One of the limiting factors during Operation Iraqi Freedom was the competition for bandwidth. While connectivity and throughput are constantly improving, demand is increasing at an even greater rate. Seemingly no matter how much bandwidth is available, there will always be a need for more. As passing large amounts of data around the network becomes commonplace, bandwidth is likely to remain a limiting factor. It is imperative that early Web-based tactical systems that use HTTP are considered complementary assets with dedicated and proven secure communications continuing to carry the data transfer load. The superior compression and performance of binary XML (exemplified by XSBC) defuses this potential "showstopper" issue for XML deployment.

The second concern with passing large amount of data over HTTP is with security. While most tactical links are well protected, HTTP security is still a developing science. Passing any sensitive data over the NIRPNET, which overlaps with the common big internet, unmistakably carries a large security risk. The SIPRNET, however, is considered to be a secure network as there are no "hard" connections the World Wide Web and its hackers. However, the SIPRNET is not under full government control in the sense that it utilizes off-the-shelf technology and runs on commercially controlled hardware.

With respect to the above considerations, this thesis strives to describe a net-centric Web-based architecture for integrating sonar modeling assets and sonar visualization tools into TDAs. To accomplish this, four requirements were outlined. The first requirement is to provide the capability to access large oceanographic analysis resources using Web services to obtain the relevant environmental inputs needed for accurate sonar modeling. Second, sonar prediction models must be exposed over the Web in such a way that each model can understand a common stream of environmental data and generate meaningful sonar prediction results that conform to particular standard. Third, a visualization tool must be developed that can parse the sonar prediction results regardless of which model was accessed and visualize the data in a standardized way. The final requirement (albeit the first to be accomplished) is to create and agree upon the standard language restrictions that will govern data transfer between the nodes of the architecture. The resulting model is illustrated in Figure 7.
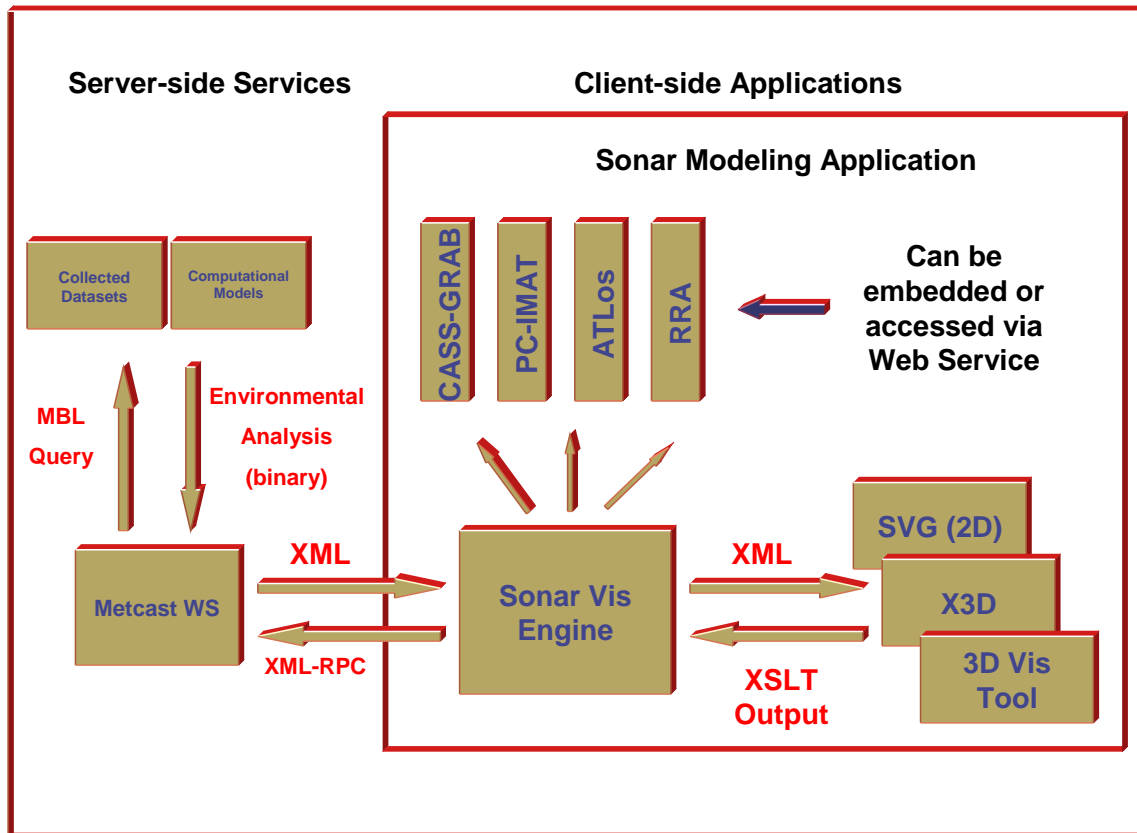

Figure 7.   Net-centric Sonar Modeling Architecture is a three-tiered approach that modularizes the data-importation, sonar prediction modeling, and visualization aspects of the sonar problem.

35

The environmental, modeling, and visualization "arms" of the model are the constants. The "body" of the model is what will determine the final realization of the decision aid. For example, the initial exemplar of the model was the AUV Workbench. Here, a Java panel with an embedded Xj3D window and swing GUI allow the user to access FnMOC's Metcast server, manually select which sonar model to run that data on, and use a local visualization module to generate X3D representations of that data that were viewable in the Xj3D window. In a separate implementation, the WEAVER project developed a web server that allows a user to make tactical inputs on a SIPRNET web site form. The web server then bears responsibility to access the oceanographic data via the Metcast server, run it through a prediction model, and run visualization code that created scenes viewable on the web page.

C.   METCAST SERVICE COMPONENT OVERVIEW

The first step towards implementing a net-centric sonar modeling architecture is to determine what oceanographic data to use as model inputs. A traditional approach is to maintain an archive of historical data that contains pertinent parameters organized by date and geographic location. In some cases, observed data may be able to be submitted as inputs. The disadvantage with this method is that it requires a rather significant resource commitment on the client side. In an ideal net-centric implementation, it is important to avoid mandating large resource commitments on the client side in order to preserve the flexibility in design implementation. Thus, the most versatile implementation would need to access the appropriate data remotely.

One of the services provided by FnMOC is a Web-enabled Metcast server. Metcast is a communication system that takes HTTP post requests and returns formatted products. FnMOC populates its Metcast server with environmental products such as sound speed profiles, sea temperatures, and bathymetry data. While this is freely available over the Web, it is not particularly well suited for a true net-centric implementation. The first problem is that, although it is accessed via HTTP, the text requesting the data must be written in Metcast Brokering Language (MBL). The second difficulty is that the data is returned in NetCDF binary format. Although a common standard of data transfer, NetCDF is not as easily manipulated and understood as XML.

Both of these issues are resolved by exposing Metcast as a Web service. Not one line of code needs to be changed to accomplish this. A Web service that takes in user input and auto-generates the MBL and then translates the NetCDF response into XML alleviates the usability issues of Metcast and gains all the advantages that Web services provide. The actual implementation of the Metcast Web Service is described in Chapter VI.

D.   SONAR MODEL COMPONENT OVERVIEW

This component of the architecture is the most important implementation within the net-centric framework. Two sonar models were chosen as the initial systems to incorporate into the architecture, however, neither will reach full implementation at the publishing of this thesis. The first model used was the Recursive Ray Acoustics (RRA) model created by Dr. Ray Ziomek of NPS. RRA is described in detail in Chapter VIII, but can be summarized as a mathematical model that traces how a sonar "ray" reacts with the ocean environment through a series of time-steps. The RRA model was implemented in Java as part of the Holiday thesis (Holiday 1998). The open source code readily accepts the data retrieved from Metcast and is implemented as a local model implanted in the AUV Workbench. The RRA is also easily exposed as a Web service. However, due to its ray tracing nature, it does not generate output that is easily converted to the schema proposed in this thesis nor is it able to generate common sonar plots as it currently stands. As will be outlined in the Future Work chapter of this thesis, the RRA code will have to be further modified to include picking algorithms in order to fully comply with the architecture proposed in this thesis.

The second model that has been somewhat integrated into this architecture is CASS/GRAB as part of the WEAVER project. CASS/GRAB is a popular sonar model software package that is used in formally-adopted decision aids such as PC-IMAT. As part of the WEAVER project, Sonalysts Inc. developed an XML "wrapper," or interface, that is able to accept environmental inputs from MWS and generate XML output that conforms "in spirit" with the schema presented in this thesis. Sonalysts and NPS are developing a formal schema that will eventually be submitted to the Undersea Warfare XML Working Group. Although the data format and layout are expected to be similar to what is proposed in this thesis, output received from the CASS/GRAB server is currently

37

separated into files by depth as opposed to being integrated into one comprehensive XML document. Also, the XML elements used to describe the data are matched with the parameter fields in CASS/GRAB. This is done as a prudent developmental step but will eventually need to be converted into more model-independent terms. In addition, CASS/GRAB is not yet fully exposed on the SIPRNET for widespread independent use.

Although neither CASS/GRAB nor RRA is completely integrated into this architecture, most of the heavy lifting has already been done and current implementations provide a valid proof of concept. As the WEAVER project progresses and the NAVAIR/Sonalysts/NPS team continues to further research this area, multiple models are expected to reach full integration into the net-centric sonar visualization environment.

E.   VISUALIZATION COMPONENT OVERVIEW

Once the output is returned from the sonar model engine, the next challenge is to represent that data in an intuitive way that is useful for training and/or decision-making. The visualizations should be standardized in both content and format. By standardizing the content independent of which model the sonar model was used to generate the prediction data, operators can then familiarize themselves with the representations and easily interpret them. It also is important to standardize the format in which the representations are generated. These visualizations should be able to be viewed regardless of implementation. For the 2D plots, both JPG and PNG image formats are commonly accepted and easily integrated into 2D viewers or posted on the web. XML-based scalable vector graphics (SVG) drawings are also increasingly prevalent. 3D visualizations can be very intuitive but are rarely available in a cross-application format. However, creating a toolkit that autogenerates sonar visualization scenes in X3D may be the solution. X3D is XML-based, which means it is self-describing and easily ported using SOAP or other XML transporters. X3D scenes can be easily converted into VRML97 which can be viewed in common browsers such as Internet Explorer or can be viewed directly by embeddable X3D viewers created by the open source Xj3D API.

The initial representations created in this thesis were designed to replicate common sonar modeling visualizations. This was done for two reasons. First, this is what

fleet operators are used to seeing. Second, in so far as they are able, they are useful visualizations. There is no need to replace the sonar visualizations present in today's decision aids. That said, the capabilities inherent in X3D provide an opportunity to improve upon these visualizations and infuse interactive features that may make the scene more useful. Given the wide number of 2D representations needed to cumulatively render sonar information, and the relatively small number of understandable 3D representations available, this approach has greater potential for revealing new capabilities.

F.   SCHEMAS

XML Schemas are formal templates that define specific XML documents and describe how data is represented for an explicit purpose. XML documents are validated against a schema to ensure they are well-formed and conform to specific application expectations. Without schemas or document type definitions (the predecessor to schemas), a client is unlikely to know how to properly form an XML document that can be understood by a server application and vice versa. Schemas provide the document-specific standards that describe how specific XML tagsets ought to be formed. The more general and supported a schema is, the more powerful it becomes.

In the research done in this thesis, three separate, native schemas have been created to manage XML: the Sonar Modeling Schema (SMS), the Metcast Response Schema (MRS), and the Sonar Source Properties Schema (SSPS). The SMS is a continually evolving "master" schema that is designed to govern data representation throughout the entire modeling process. As previously unaccounted for resources are included into the architecture, this schema will be amended to accommodate them. The MRS specifies the XML stream returned by MWS. It defines how each analysis product returned by the Metcast server will be represented in XML. Clients can use this schema to validate responses and develop code to manipulate it. The SSPS was developed as part of the AUV Workbench to define XML documents that describe the characteristics of various sonar sources to limit the number of inputs required of the user.

When the SMS or an alternate schema becomes an approved specification, it will become the standard for defining all sonar modeling and simulation data. When this occurs, the MRS and the SSPS will become defunct. Originally, the MRS and SSPS were designed to freely interoperate with the SMS. As this research has progressed, however, the SMS has evolved to the point where a transformation is needed to incorporate the other schemas into it. The MWS and the AUV Workbench will have to be modified to accommodate an approved governing set of schemas. Eventually, this schema might be incorporated into a schema that describes data for all undersea warfare. Likewise, the undersea warfare schemas might be used with other warfare schemas to form a comprehensive tactical combat schema, and so on. Once these schemas are accepted and approved as a standard, all systems that use them will conform to the same set of schema and, as a result, all these systems can "speak" the same language by exchanging meaningful data interoperably.

1.   Sonar Modeling Schema (SMS)

The development of the SMS represents one of primary goals of this thesis. Unfortunately, the schema cannot be fully defined without a robust "roundtable" discussion between the major sonar modeling principles – a difficult (but feasible) task. As part of this thesis, the SMS presented represents a small minority of the sonar modeling resources available to the fleet. The real purpose of the SMS as described in this thesis is as a starting point, or exemplar, to demonstrate the value that a "master" schema can have on sonar modeling as exhibited in the WEAVER project. Figure 8 presents the five main subelements of the sonar modeling problem. These are ScenarioDescription, ModelOutput, EnvironmentalData, SonarSource, and Target. ScenarioDescription provides the appropriate meta data needed to detail the analysis. The EnvironmentalData element defines how oceanographic data required as inputs to a sonar modeling package are to be represented. ModelOutput describes derived output from a sonar prediction model. The last two elements, SonarSource and Target define characteristics about the sonar sources and targets within the given scenario. The full SMS is offered in Appendix J.
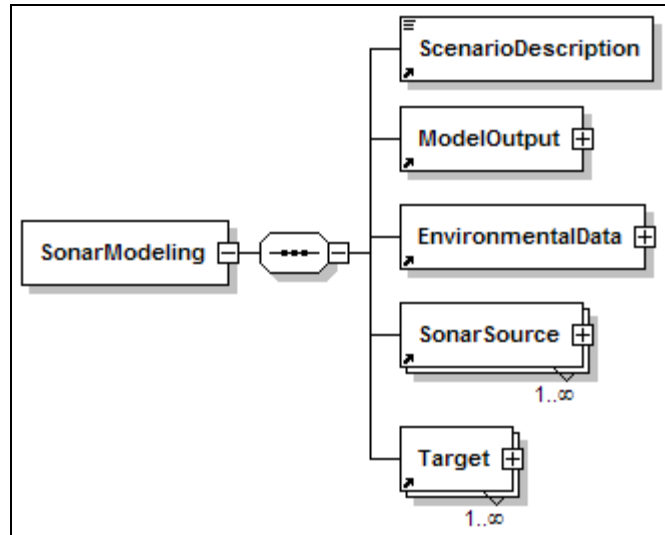
Figure 8.   The Sonar Modeling Schema's top element's five children divide the sonar problem into its main components.

As SMS continues to evolve, it will hopefully merge with other undersea modeling and decision aid schemas and eventually will become a component in a set of undersea warfare master schemas. The schema as it stands currently is designed to fulfill the needs of this thesis and support the WEAVER project. This schema is understood by code that interfaces both with the CASS/GRAB and RRA model software packages. As more sonar models become available to this project, the schema can be further adapted to incorporate the needs of those new models as well. Ideally, in the future the schema can be described in a manner understood by encapsulating interface code for all common sonar prediction models. Figure 9 shows the SonarModeling element in XML text.

```
<xs:element name="SonarModeling">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="ScenarioDescription"/>
      <xs:element ref="ModelOutput"/>
      <xs:element ref="EnvironmentalData"/>
      <xs:element ref="SonarSource" maxOccurs="unbounded"/>
      <xs:element ref="Target" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Figure 9.   Top-level elements as defined in the Sonar Modeling Schema.

The EnvironmentalData element represents the data that, whether observed or acquired through analysis, describes the ocean environment. This data is needed to run accurate sonar prediction models. Currently, this element is modeled after the MRS discussed later in this section. The EnvironmentalData subelements correspond to the products returned from MWS. These products are SoundSpeed, SeawaterTemperature, Wind, SignificantWaveHeight, and Bathymetry. Each of these elements is defined by the schema. SignificantWaveHeight, as shown in Figure 10, is defined by its attributes and element value. Required attributes are the model used to generate the data and the units the data is represented in. The schema uses enumerations to rigorously restrict what units can be used. In this case, if a unit other than feet or meters is used then the file is not valid.

```
<xs:element name="SignificantWaveHeight">
  <xs:complexType>
    <xs:simpleContent>
      <xs:extension base="xs:decimal">
        <xs:attribute name="model" type="xs:string" use="required"/>
        <xs:attribute name="units" use="required">
          <xs:simpleType>
            <xs:restriction base="xs:string">
              <xs:enumeration value="ft"/>
              <xs:enumeration value="m"/>
            </xs:restriction>
          </xs:simpleType>
        </xs:attribute>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>
```

Figure 10.   The Sonar Modeling Schema defines the SignificantWaveHeight element.

Figure 11 shows the organization of the SonarSource element. This element defines the characteristics of the sonar pulse to be modeled. Based of the Sonar Source Properties Schema, it allows a decision tool to select a particular sonar source, such a specific sonobuoy or hull sonar, and access XML-based files that associate necessary source-related input parameters to the sonar model package such as frequency and intensity.
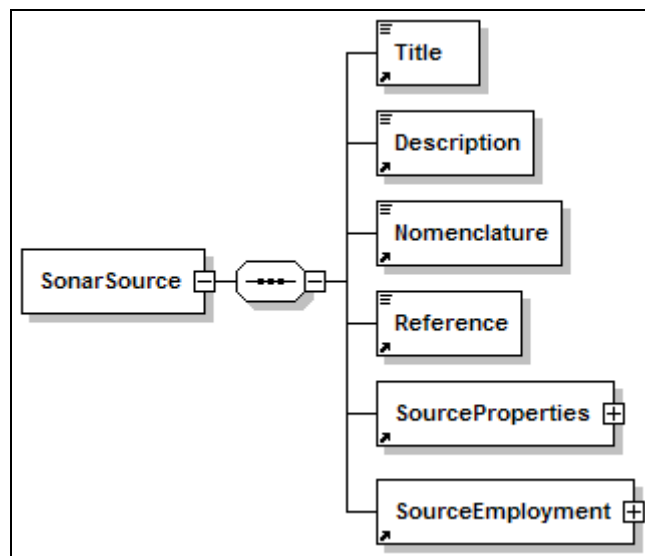

Figure 11.   The SonarSource element of the Sonar Modeling Schema describes the source and defines its properties and employment characteristics.

2. Metcast Response Schema (MRS)

The MRS was the first schema developed in this thesis. Its purpose is to govern the XML response returned from the MWS. The MRS echoes the response data retrieved by the NetCDF parsers. This was done to facilitate troubleshooting between client-side and server-side software. As the SMS becomes representative of most sonar modeling resource data structures and evolves beyond major revision, the MRS (and MWS) can be easily modified to seamlessly interoperate with SMS data.

3. Sonar Source Properties Schema (SSPS)

The SSPS was created to govern the XML files used by the Sonar Visualization Workbench Panel to import source characteristics for input into the RRA model. Instead of requiring the user to input a large number of source attributes such as frequency and beam angle via a GUI, the SVWP allows the user to select an XML file the already describes these parameters. Since these values rarely change for a particular source type, these files can be used repeatedly and the user will not be responsible for providing this data for each run. Like the MRS, this schema will require modification once the SMS reaches a consistent state. Currently, a minor transformation must occur for it to comply fully with the SMS.

G. SUMMARY

A three-tiered net-centric sonar modeling architecture appears to be the most efficient method of distributing sonar modeling assets. Three classes of assets exist: oceanographic analysis tools, sonar modeling software packages, and sonar visualization/interpretation applications. By creating a standard for the interface between these classes, assets become "plug and play" capable. That is, an application based off this architecture can heuristically choose what environmental data sources to use, what sonar modeling package to run, and what visualization tool to use without alteration because each system communicates to other systems using the same language and protocols. Implementations can vary from a heavy application to a simple web page.

V.   LEVERAGED WEB TECHNOLOGIES

A.   INTRODUCTION

This chapter describes the Web technologies utilized in the development of this thesis. These include primary APIs, toolkits, and standards that are all freely available and downloadable from the Web.

B.   JAVA WEB SERVICE DEVELOPER'S PACK (JWSDP)

The JWSDP is an open source integrated toolkit that allows a developer to build, test, and deploy XML applications, Web services, and Web applications. The JWSDP provides Java standard implementations of existing Web technologies to send and receive SOAP messages, browser and retrieve messages from Universal Description, Discovery, and Integration (UDDI) registries, as well as quickly build and deploy Web applications and servers. It does this by providing APIs for manipulating XML and SOAP such as Simple API for XML (SAX) and SOAP with Attachments API for Java (SAAJ). It also includes the Ant build tool as well as the Apache Tomcat servlet container. Information on Java and the JWSDP can be found at (http://java.sun.com).

1.   Apache Tomcat

Apache Tomcat is an open-source servlet container released by the Apache Jakarta Project. The purpose of Tomcat is to produce standard-compliant support for servlets and Java Service Pages (Chopra and others, 2003). Servlets are Java code written to dynamically generate server content and provide a means for deploying an application on the Internet. Tomcat receives requests from a client and "directs" the request to the appropriate servlet. The request is then serviced by the servlet and a response is returned to the client. In this thesis, the servlet is another Apache product that handles SOAP requests. Figure 12 shows a screenshot of the Apache Tomcat Application Manager page. The list of links indicates the servlets currently installed in the servlet container. Information of Apache program can be found at (http://www.apache.org). This thesis utilized Apache Tomcat 4.1.29.
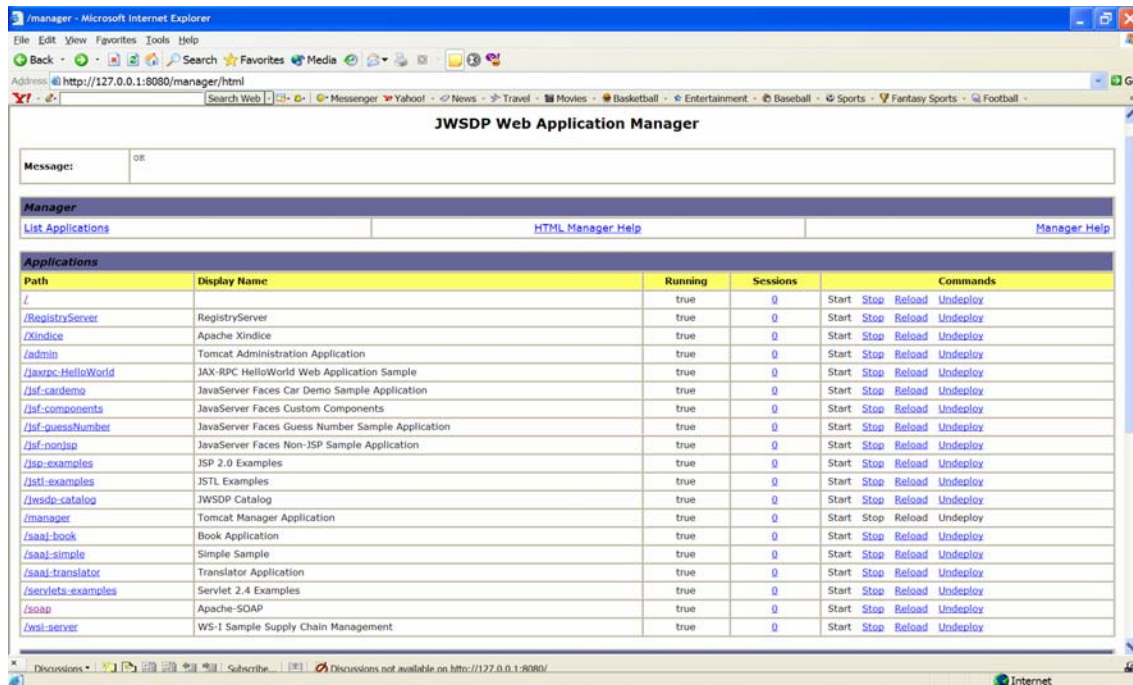
Figure 12.   Apache Tomcat Web Application Manager allows the user to deploy servlets
            to be hosted by the Tomcat server.

2.   SOAP with Attachments API for Java (SAAJ)

The SAAJ provides a standards-based way to send XML documents over the
Internet using SOAP. A SAAJ-based client sends point-to-point messages directly to a
Web service in a request-response format. A SAAJ-based client instantiates a
SOAPConnection object that handles the communication between the client and server.
The SOAPConnection call method sends the SOAP request message and then blocks
until it receives a response (Cerami, 2002).

3.   Simple API for XML (SAX)

SAX is an event-driven, serial-access mechanism for accessing XML documents.
SAX was chosen for the work done in this thesis because it's the fastest, least memory-
intensive, and the parsing requirements in this project do not require the object modeling
features some of the other parsing APIs such as Document Object Model (DOM) offer.
The SAX protocol, however, does require more programming than DOM. Since it is
event-driven, the programmer must define callback methods that are invoked as specific
elements are read. Another disadvantage is that it is a serial parser. Thus, the parser

cannot return to an earlier part of the document or rearrange it in any way. This is not a requirement for this thesis, so the SAX parser was chosen for its efficiency advantages and its flexibility. A DOM or different parser (such as JDOM) might be substituted (Cerami, 2002).

C.   WEB SERVICE TECHNOLOGIES

A web service is any service that is available over the Internet that uses standardized XML messaging system and is not tied to any one operating system or programming language (Cerami, 2002). Through Web services, diverse applications can discover each other and exchange data over the Web. Because it makes use of the XML standard, Web services allow computers to communicate with each other regardless of form or function. This capability opens the door to a whole new array of functionality to legacy systems that were previously thought to be dead ends, i.e. too expensive or too difficult to upgrade.

1.   Extensible Markup Language (XML)

XML is a text format based on a simple set of rules that allows structuring data by creating tags and attributes. Because of its self-describing nature, XML documents are extensible, independent of language and platform, and support internationalization and localization. XML was designed as a subset of Standard Generalized Markup Language (SGML) which is a text-based language used to markup (i.e. describe) data. It looks quite a bit like Hyper-Text Markup Language (HTML), another descendant of SGML, but adds composability, extensibility, and is self-describing. Unlike HTML, XML leaves the interpretation of the data it contains completely up to the application that reads it (http://www.w3.org/XML/1999/XML-in-10-points). Thus, one XML document containing oceanographic data can be used to visually model ocean environments or as inputs into a sonar model prediction engine.

XML is license-free and well-supported in the business world. An ever growing number of technologies exist to provide useful services that can manipulate XML. Extensible Style Sheets (XSL), Extensible Stylesheet Language for Transformation

(XSLT), DOM and many open-source APIs exist that allow users to easily accomplish important tasks to read, display, or manipulate XML and its inherent data. Since any XML document can be read by any XML parser, regardless of application or platform, XML can make data universally accessible. It is extensible because a user can shape the data in an arbitrary manner. In fact, XML is not really a language per se, but instead a consistent, meta-language approach for creating languages that can do different things. With this in mind, a user can create a schema that describes how a particular data set is defined. Any XML document that can be validated against that schema can be understood by any application set up to be compatible with that schema. So, for example, if an undersea warfare schema is developed and defined as a specification, then any conforming undersea warfare system regardless of platform or language will be able to understand that data.

The Department of the Navy XML policy was signed by the Chief Information Officer (CIO) Don Wennergren in December 2002. The policy sets parameters for how the Navy will use XML and provides direction to help officials manage areas critical to successful XML implementation. The XML Functional Namespace Coordinators (FNC) is responsible for maintaining uniformity in Navy XML vocabularies. The XML policy is part of a comprehensive Navy-wide XML implementation strategy.

2. Simple Object Access Protocol (SOAP)

Simple Object Access Protocol (SOAP) is an XML-based set of rules for exchanging information between computers. The SOAP specification defines a simple set of rules for translating platform specific data types that ordinarily would not be compatible with any other system into XML, which is platform and language independent (Cerami, 2002). SOAP also defines an XML-based envelope to transfer this information. Because SOAP is XML-based, it operates independent of platform and language. Thus, heterogeneous clients and servers can become interoperable. One of the main features of SOAP is the ability to invoke remote procedure calls (RPC) using Hyper-Text Transfer Protocol (HTTP). RPC's are the basis of distributed computing. This allows a Java client installed on a laptop aboard a ship at sea to invoke methods

resident on a land-based Linux supercomputer by sending SOAP packets over the internet. Such flexible capability is the cornerstone of a net-centric architecture.

3. Apache SOAP / Apache Axis

Apache SOAP is an open-source Java implementation of the SOAP protocol. Apache SOAP installs as a servlet application in Tomcat. Once installed, the Apache SOAP Admin feature allows the programmer to configure SOAP Web services via a Web form. Once deployed, the Tomcat server receives incoming requests from a client and forwards them to the Apache rpc-router servlet. The rpc-router then looks up the requested SOAP service, instantiates an object of the servlet class, and invokes the specific method to accomplish the desired task.

Figure 13.   The Apache SOAP web form walks a developer through the process of deploying a SOAP web service.

The TCP Tunnel GUI tool is a valuable programming resource in the Apache SOAP package. When deployed, the tool listens for SOAP requests on a specified port and then forwards the SOAP message on to its destination. This effectively creates a "tunnel" between two network ports. As the Tunnel tool receives SOAP traffic, it displays the messages as they are passed back and forth in a GUI window. This ability to

monitor the actual SOAP conversation between the client and server can aid in understanding SOAP as well as debugging live applications. Figure 14 is a screenshot of the TCP Tunnel GUI displaying a MWS request-response sequence.
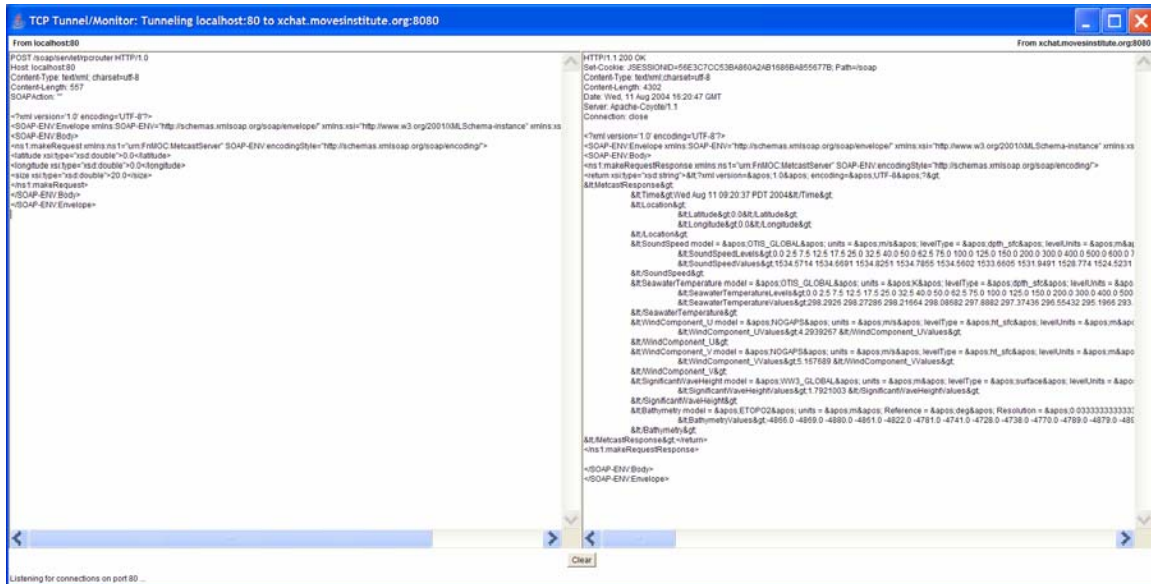
Figure 14. To aid debugging, the TCP Tunnel/Monitor allows a developer to view SOAP messages as they are sent over the network (Cerami, 2002). The left window shows query inputs, while the right window shows corresponding service responses.

Apache Axis is the follow-on project to the Apache SOAP application. While Apache SOAP was used for most of the original research done in this thesis, future work will be deployed using Apache Axis. Proven as a reliable and stable base, Apache Axis 1.1 was used to host the latest version of MWS on the SIPRNET in support of the WEAVER Project. Apache Axis is also open source and freely downloadable. It is a complete re-architecture of the Apache SOAP implementation and has many new features that expand its Web service capability (Cerami, 2002).

D.   XJ3D

Xj3D is a project of the Web3D Consortium Source Code Working Group that provides an open-source Java toolkit for viewing VRML97 and X3D scenes. While Xj3D hasn't reached version 1.0 release status as of this thesis, it does provide a stable open

50

source API that implements a browser that is able to natively view 3D content. Written entirely in Java and openly published, it is easily incorporated into research applications that require display of 3D data visualization scenes or just ordinary browsing windows. Information on Xj3D can be found at (http://www.xj3d.org).

E.   X3D

X3D is an XML-based three-dimensional (3D) open file format that enables real-time communication of 3D data across network applications. X3D is easily integrated with web services and distributed networks. The X3D specification has been approved by the International Standards Organization and supports open architecture design, provide for 3D free-viewpoint navigation, and is compatible with most common internet browsers. It provides an XML-encoded scene graph that describes a 3D virtual world. This scene graph is easily translated into VRML97 and viewable in most browsers with a plug-in or displayed directly in an Xj3D browser. These scenes are navigable, interactive, and real-time. Information on X3D can be found at (http://web3d.org/x3d).

F.   CORTONA VRML CLIENT

Cortona VRML Client is a browser plug-in that is excellent for viewing 3D content on the web. It contains a set of optimized 3D renderers and allows supported Internet browsers to display interactive 3D scenes written in VRML97. Cortona was developed by Parallel Graphics and is freely downloadable off the web. Cortona was chosen for this thesis because it features complete VRML97 support, supports advanced rendering techniques, and takes advantage of 3D graphic accelerators. Information on Cortona can be found at (http://www.parallelgraphics.com/products/cortona).

G.   JFREECHART

JFreeChart is an open source Java API created by David Gilbert and released under the GNU Lesser General Public License. The JFreeChart API is freely downloadable over the web. It allows a programmer to pass in arrays of data and create 2D charts depicting that data and saves them as images that conform to Joint Picture Experts Group (JPEG) or Portable Network Graphics (PNG) formats. These images can

then be imported in X3D scenes or displayed on a web. Information on JFreeChart can be found at (http://www.jfree.org/jfreechart).

H.   HTTPCLIENT

HTTPClient is an open-source Java API created by Innovation Gmbh and released under the GNU Lesser General Public License. The HTTPClient API provides a complete library to efficiently communicate over HTTP using various methods such as head, get, post, and put. In this thesis, the HTTPClient API is utilized by MWS to communicate to the Metcast Channels server over HTTP. HTTPClient methods post a properly formatted query to the Metcast web server and retrieve the response for client use. Information on HTTPClient can be found on Innovation Gmbh's website (http://www.innovation.ch/java/HTTPClient).

I.   NETWORK COMMON DATA FORM (NETCDF)

The NetCDF Java Library is a Java-based API designed to act as an interface for array-oriented data in NetCDF format. It was created by the Unidata Program Center and released under the GNU Lesser General Public License. The NetCDF libraries define a self-describing, machine-independent format for representing scientific data. The NetCDF design serves as a portable, efficient file format and programming interface for storing and retrieving NetCDF files across multiple platforms (Li and others, 2003). It provides scientific applications with a common data access method for storage of structured datasets. The advantages of NetCDF are that it allows for direct access to subsets of large datasets, it is sharable, and data can be appended to a dataset without having to redefine its structure. Information on NetCDF can be found at (Http://www.unidata.ucar.edu/packages/netcdf).

J.   SUMMARY

Many technologies were leveraged in the research completed in this thesis. Java technologies such as JWSDP, SAX, and SAAJ were the starting point for much of this work. Apache Project releases of Tomcat and Apache SOAP was also vital to this project's success. In addition, many other technologies such as HTTPClient, JFreeChart,

X3D, Xj3D, Cortona and NetCDF helped perform specific tasks needed to demonstrate the theories outlined in this thesis.

THIS PAGE INTENTIONALLY LEFT BLANK

VI. METCAST WEB SERVICE (MWS)

A. INTRODUCTION

The chapter describes the Metcast Web Service (MWS). MWS was the first step taken in this research towards implementing the sonar modeling framework. Before sonar modeling can occur, environmental data must be obtained. This chapter presents the motivation behind MWS development, the framework that governs it, the original Metcast Channels server, and MWS itself.

B. MOTIVATION

Sonar prediction models are only as good as the environmental data that the model ingests. FnMOC provides an extremely valuable resource to the fleet by populating large data bases with high-resolution environmental analysis data. This data includes sound speeds profiles, sea temperatures at various depths, and many other oceanographic parameters that are vital for accurate sonar prediction. FnMOC provides this data via a web server interface called Metcast. Metcast is a communication tool that allows systems to distribute products over the Web by accepting requests via a HTTP post. While Metcast provides a service over the Web, it is not a Web service. Instead, the post must be written in a distinct language called Metcast Brokering Language (MBL) that is not yet in common usage. The Metcast server then returns the requested data back over the web in binary NetCDF format. While NetCDF is an established data format, it is not as widely-supported as XML.

While this is extremely useful data provided over a freely accessible medium, there are still several obstacles a user must overcome in order to use these resources. First, a user must spend time learning a stovepipe, or application-dependent, language called the Metcast Brokering Language (MBL) in order to make the correct requests to the server. The user must have prior knowledge of where the server is located on the web, how to access it, and how to parse the response. The second disadvantage to the Metcast system is that it returns the data in NetCDF binary format. Again, this is satisfactory when the user is familiar with Metcast and NetCDF, but this is not in keeping with an

ideal net-centric architecture that allows any potential user to automatically find the web service, automatically determining how to access it and how to parse the returned data. It will be advantageous to the fleet and be in keeping with the net-centric architecture it desires if Metcast were exposed in such a manner that new and legacy applications, regardless of platform or language, could easily gain access to that data.

C.   PROPOSED FRAMEWORK

The goals outlined above can be accomplished by converting Metcast into a true Web service. The Web service will act as a translator that converts Web service XML-based requests into a well-formed MBL message that can properly access the Metcast server. The Web service then converts the NetCDF response from Metcast into an easily parsed XML stream that is returned to the client. In this design, a client only needs to be able to handle XML as inputs and outputs and does not need to worry about having to decipher MBL or NetCDF. On the server side, no changes need to be made to the Metcast service code which never comes into contact with XML. MWS was developed to accomplish this as part of this thesis with assistance from FnMOC.

MWS allows a user to make remote procedure calls from a client over the Web to MWS via XML using SOAP as the transport mechanism. An initial implementation allows the user to simply specify the position and size of the geographic area that the user wishes to obtain data about. MWS then acts as a broker and makes the appropriate MBL request to Metcast. The Metcast system returns the resultant data in binary NetCDF form which the MWS uses code developed by FnMOC to parse the binary data stream. Code modifications made in conjunction with this thesis insert the data into an XML string. This string is then returned to the client in this much more manageable form.
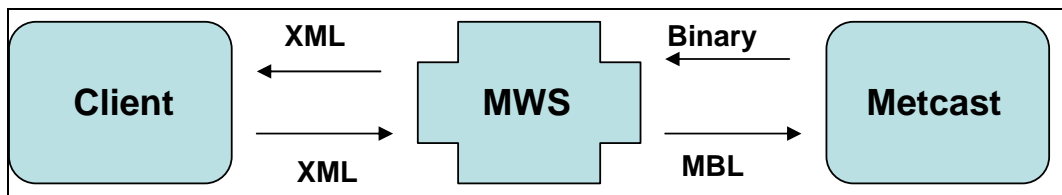
Figure 15.   The Metcast Web Services acts as the interface with the Metcast server by translating an XML-based request from the client to Metcast's native brokering language and then transforming the returned binary NetCDF data into XML.

By converting Metcast into a true web service, its description can be published on Web registries called Universal Description, Discovery and Integration (UDDI). The UDDI will register the MWS through a Web Service Description Language (WSDL) file which specifies exactly what parameters and methods are needed to retrieve the appropriate data. Technology exists that can automatically find and access the web service. In addition, open-source software code is freely available for users to access and parse the data using various XML-based APIs. Clients may use this data to create 2D or 3D visualizations of the oceanographic data itself, as inputs to high-performance sonar prediction models, or any number of tactical purposes.

The initial implementation of MWS is designed to return environmental data specific to the RRA sonar model with intention of expanding to accommodate follow-on models as they become exposed on the Web and their frameworks become known. The products returned are sound speeds at specified depths, sea temperatures at specified depths, wind speeds (provided in U and V vector components), significant wave height, and bathymetry depths. These are all explained in more detail in the next section. The CASS/GRAB model to be implemented in the WEAVER project requires full Modular Ocean Data Assimilation System (MODAS) data as opposed to simple sound speed data and will be added in future versions.

MWS was initially made available on the Non-secure Internet Protocol Router Network (NIPRNET). JWSDP 1.3 is used to deploy the service on a NPS server (http://xchat.movesinstitute.org:8080). JWSDP includes the Tomcat 4.1.29 servlet container which is used to listen and process Web service requests. In order to use the

SOAP, a SOAP-specific handler must be installed on top of Tomcat. Apache SOAP was the initial handler used for MWS. Later versions of MWS use Apache Axis 1.1. MWS was successfully tested using a client developed inside the AUV Workbench. For tactical purposes, MWS was made available on the SIPRNET on a FnMOC Linux server and was successfully tested during the WEAVER project demonstration during the Rim of the Pacific (RIMPAC) 2004 exercise.

D. WHAT IS METCAST?

1. Meteorology Forecast (Metcast) Channels

Metcast is a request-response communication system designed to distribute weather information over the Web. A Metcast server maintains a database of weather observation reports, forecasts, weather model analysis, and many other real-time information products. A Web server then delivers a subset of the data in response to a client-based query.   A Metcast client submits a request for one or several products via HTTP protocol. The request must be formulated in its native Metcast Brokering Language (MBL). The client can be a web-form or an application.

2. Metcast Products

MWS calls upon the Metcast service to provide sound speed and sea temperature as a function of depth, as well as wind components, significant wave height and bathymetry data. As explained in the previous section, Metcast is not an oceanographic model itself but just a mechanism for distributing products. Metcast retrieves these products from a data base that is populated by U.S. Navy approved environmental models. Models descriptions follow.

The Optimum Thermal Interpolation System (OTIS) Global model is an optimum interpolation based objective analysis scheme designed to produce ocean temperature analysis (https://www.fnmoc.navy.mil/PUBLIC). OTIS uses observed inputs from ships, buoys, and satellites to generate results. The Global version of OTIS, which is non-eddy resolving, produces output analysis every 12 hours at low resolution (1.0 degree) and every 24 hours at high resolution (0.25 degree). Temperatures are generated at 34 vertical levels representing the upper 5000 meters of the ocean.

Wind speed and direction is provided by through the Navy Operational Global Atmosphere Prediction System (NOGAPS). The NOGAPS forecast model is a global model that uses satellite-derived observations to predict low level winds, precipitation and tropical storm tracks (https://www.fnmoc.navy.mil/PUBLIC). Wind speed and direction are returned as vector components U and V. The U-component represents the velocity of the wind from west to east in meters-per-second, while the V-component represents the north-south component. Straightforward trigonometric calculations on the client side can then determine wind speed and direction. NOGAPS runs four analyses per day and can provide forecasts out to 144 hours.

The Wave Watch III (WW3) Global model generates the significant wave height values returned by Metcast. WW3 is a third generation wave model developed at the National Oceanic and Atmospheric Administration (NOAA). WW3 forecasts the evolution of directional wave energy spectra, from which significant wave height, wave period and wave direction fields are generated (https://www.fnmoc.navy.mil/PUBLIC). WW3 Global uses a coarse-resolution configuration, but still includes shallow water physics to account for refraction and bottom friction events. MWS provides WW3's significant wave height as a product, which represents the average height of the highest third of the waves in a wave spectrum. Significant wave height can be directly converted into sea state. WW3 runs twice per day and can provide forecasts out to 144 hours.

The Global 2 Elevation (ETOPO2) system is a global elevation data base generated by NOAA. ETOPO2 is set at two-minute resolution (latitude-longitude) and provides bathymetry data in meters for the entire world. ETOPO2 is really a compilation of five topographical data bases that were re-gridded to two-minute spacing for uniformity that covers both land and the sea floor (https://www.fnmoc.navy.mil/PUBLIC). As the preferred product input for sonar prediction models such as CASS-GRAB, Modular Ocean Data Assimilation System (MODAS) will eventually replace OTIS as the primary model for sound speed and sea temperature analysis. MODAS is a modular toolkit for conducting analysis on ocean conditions. It consists of over 100 individual programs that acquire input data from various sources to predict present conditions in the ocean to include sea temperature and salinity (https://www.fnmoc.navy.mil/PUBLIC). These

sources include climatology, observed sea surface temperatures, and in-situ measurements.

E.   METCAST WEB SERVICE (MWS)

1.   The Metcast Response

The response returned by MWS is governed by the Metcast Response Schema (MRS) is described in Chapter IV. MRS maps directly to the MetcastResponse element of the Sonar Modeling Schema (SMS) and will eventually validate against this master schema. In general, the environmental data is described by format, units, and the analysis model used to develop them. The excerpt below shows the returned XML for a sound speed profile element. From its attributes, a user can determine that the sound speed values are measured in meters per second, that the levels (depths) are specified in meters, and that the OTIS Global analysis model was used to generate the data.

```
<SoundSpeed model="OTIS_GLOBAL" units="m/s" levelType="dpth_sfc" levelUnits="m">
   <SoundSpeedLevels>2.5 7.5 12.5 </SoundSpeedLevels>
   <SoundSpeedValues>1535.2701 1535.3052 1535.3455 </SoundSpeedValues>
</SoundSpeed>
```

Figure 16.   MetcastResponse schema element as returned by MWS includes a string-bases array of depths (levels) and speeds (values).

For each request, the time of request, latitude, longitude, and size of area are mandatory elements for the document to be well-formed. All other fields are optional. This is because values for a particular data element are occasionally not available on the server. However, the remainder of the XML document is still useful and thus still expected to be valid and conform to the schema. When a data element is missing, such as sound speed profiles, then the client may choose to use historical data in its place.

The analysis products returned from MWS are sound speed profiles, sea temperature at specified depths, wind vector, significant wave height and bathymetry at two nautical mile increments. The sound speed profiles (SSPs) and bathymetry data feed directly into the RRA and CASS/GRAB sonar model algorithms. The sea temperatures, wind speed and significant wave height data are not yet imported directly into the modeling equations, but still carry great values as situational awareness and training tools

60

when properly visualized. Figure 17 shows a top-level view of the MetcastResponse element of the SMS.
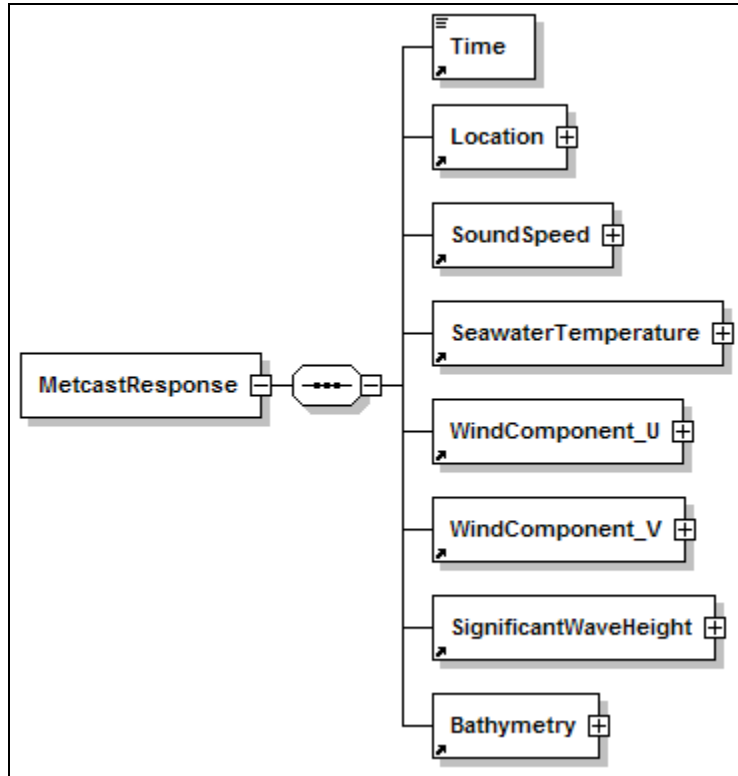
Figure 17.   MetcastResponse schema element defines the response returned by the Metcast Web Service.

The sound speed profiles, the sea temperatures, and bathymetry products are stored in Metcast as an array (or arrays) of doubles. For these elements, it would be inefficient to create tags and indices for each value as the arrays can become quite large and create significant overhead by declaring XML elements for each array value. Instead, it is more efficient to pass them as arrays of doubles, written out as string values. This may seem to burden the user by forcing the client application to parse the string into an array for data manipulation. However, converting the string into an array is a straightforward endeavor and is an inherent capability in the EnvironmentalData API described in the next chapter. In addition, it provides for a direct transition to document-fed sonar models that require a text-based input file. Elements that require a series of

values in relation to a paired parameter, such as sea temperature at specified depths, are described by a pair of sub-element strings. Finally, such representations allow strong type checking for valid data by using the corresponding schema.

For example, in the figure below the SeaTemperature element is comprised of a SeaTempLevels string and a SeaTempValues string. SeaTempLevels specifies the depths and SeaTempValues depicts the temperatures themselves. This is organized so a parser can read in each string, convert them independently into arrays of equal size, and bind them together in a native programming language using a grouping construct or a separate object. In addition, an open source API for parsing the returned XML was developed as part of this thesis and is described in detail in the next chapter.
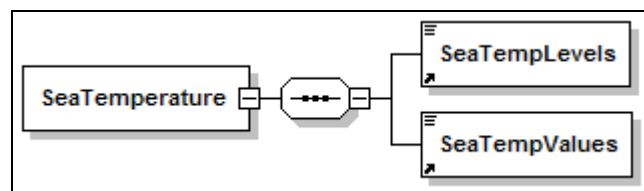
Figure 18.   SeaTemperature schema element defines the parameters needed to describe temperature as a function of depth.

### 2.	Source Code Implementation

The MWS was developed through a coordinated effort between NPS and FnMOC. The effort began when FnMOC provided NPS with a sample Metcast client in the form of Java source code. The FnMOC code made use of two open source APIs called HTTPClient and Java NetCDF, each made available under the GNU Lesser General Public License for open-source. HTTPClient provides methods to efficiently conduct HTTP communication and the Java NetCDF API parses files in NetCDF format. The sample code created a well-formed MBL message and delivered a HTTP post to an exposed FnMOC Metcast server. The code then uses the Java NetCDF API to parse the returned NetCDF document into distinct Java objects for the client's use. In the sample code, the client simply outputted the results as text to a console window. For the MWS, the client becomes an intermediary and acts as a server for the true client. This true client can be a sonar model, a decision aid, or a visualization tool.

To modify the sample code to meet the requirements outlined for the MWS, two modifications were made. First, the code is altered to allow a user to dynamically select the size and location of the region to be requested. Second, the code is redesigned to return the Metcast data encoded in XML. This was achieved by allowing a client to call one specific method (via a Web service) and set the parameters needed to make the desired Metcast call, and also set the requested environmental data as the return value as an XML string. Before it is returned, the Metcast data is parsed as NetCDF and then reformatted as XML. The finished code was then exposed as a Web service by installing it in a Tomcat Web servlet container directory. The servlet container accesses a particular Web service by its deployment descriptor. The deployment descriptor provides information regarding the physical location of the code in relation to the server, its Uniform Resource Name (URN), and describes the methods that can be accessed by a remote procedure call. Once exposed, the service is accessed by a client using an open source Java API for SOAP. The string can then be written to a file or streamed directly to a parser for processing.

F.   SUMMARY

MWS is the interface that links Metcast environmental data needed for effective sonar modeling to the sonar modeling framework. By acting as a translator, MWS transforms relatively unfriendly languages used by Metcast (NetCDF and MBL) into XML for general use over the network as Web services.

THIS PAGE INTENTIONALLY LEFT BLANK

VII. EXAMPLE IMPLEMENTATIONS

A. INTRODUCTION

This chapter provides an example implementation of the net-centric sonar architecture. This implementation originally was developed as a standalone client that used Metcast data to run RRA code in an Xj3D browser. Eventually, this involved into a panel that uniquely visualizes model results using the Sonar-Visualization API. After sufficient testing, it was finally incorporated into the AUV Workbench.

B. METCAST CLIENT

1. SOAP Client

After the MWS was deployed on the Web, the next logical step in the development process is to develop a client to access it. Initially, a single-class client was written to access MWS and output the returned XML to the console. After initial testing proved satisfactory, this class was used as a basis for creating a capable sonar visualization client. The first implementation was a stand-alone Java swing JPanel that allows a user to formulate a Metcast query by manipulating a GUI, run a local RRA model using the data received in the Metcast query, and then visualize the data in an embedded Xj3D browser. Eventually, the panel migrated into the AUV Workbench as an initial tactical application implementation.

With only slight modification, the same class that interfaced with MWS is used in the Sonar Visualization Workbench Panel as well as in the WEAVER Project. The class makes a simple SOAP XML-RPC call to MWS and listens for the response. Figure 19 is the outgoing SOAP XML-RPC message from the client to the server as captured by the TCP Tunnel/Monitor feature of JWSDP-1.3.

```
POST /soap/servlet/rpcrouter HTTP/1.0
Host: localhost:8080
Content-Type: text/xml; charset=utf-8
Content-Length: 561
SOAPAction: ""
<?xml version='1.0' encoding='UTF-8'?>
<SOAP-ENV:Envelope
    xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <SOAP-ENV:Body>
        <ns1:makeRequest xmlns:ns1="urn:FnMOC:MetcastServer"
            SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/>
            <latitude xsi:type="xsd:double">-15.0</latitude>
            <longitude xsi:type="xsd:double">-145.0</longitude>
            <size xsi:type="xsd:double">1.0</size>
        </ns1:makeRequest>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Figure 19.   Outgoing SOAP request to the Metcast Web Service as captured by the
             Apache SOAP TCP Tunnel GUI in Figure 14.

The first line is a standard HTTP post signature that tells the server that it is of
post form and that the message should be directed to the RPC router at the given location.
The next few lines specify what port the host in listening on (in this example, the Web
service was located on the same machine as the client) and the length and type of the
message. The remaining part of the post is the XML itself. It consists of an XML header
and the SOAP envelope. The SOAP Envelope contains header information delineating
what schemas are in use.

The SOAP Body, as part of the Envelope, contains the payload. For an XML-RPC
request, the payload consists of what service is to be called depicted by the URN, the
method name (in this case, appropriately named "makeRequest"), and fields that describe
the values that will be passed in as parameters of that method. In this example, they are
three doubles describing latitude, longitude, and size. When this request is received by
the RPC router, the appropriate method is called on the servlet inputting these
parameters. The response is of similar format with the exception that the method element
is replaced by a return element. The return element consists of the return value of the
invoked method. In MWS, the "makeRequest" method returns a string that is already in
XML form.

This request is generated by use of the open source SOAP API created as part of the Apache's Web Services Project. The call itself is invoked by only a few lines of code. The following is an excerpt from MetcastTestClient class as implemented in the AUV Workbench that resulted in the above SOAP message.

```
// set up call
Call call = new Call();
call.setTargetObjectURI(TARGET);
call.setEncodingStyleURI(Constants.NS_URI_SOAP_ENC);

// describe method and its parameters then bind to variables
call.setMethodName("makeRequest");
Vector params = new Vector();
params.addElement(new Parameter("latitude", Double.class, lat, null));
params.addElement(new Parameter("longitude", Double.class, lon, null));
params.addElement(new Parameter("size", Double.class, size, null));
call.setParams(params);

// make call
org.apache.soap.rpc.Response resp = call.invoke(url, "" );
```

Figure 20.   Java code excerpt from MetcastTestClient.java (Appendix D) that invokes a SOAP request to MWS.

The three parameters that are passed in to the "makeRequest" method represent the latitude and longitude of the north-west corner of the box to be analyzed and the length in nautical miles of one of its sides. The "makeRequest" method makes the RPC call to MWS and returns a string that is an XML representation of the resultant oceanographic data. In the original implementation, this consisted of sound speed profile, sea temperature, wind vectors, significant wave height and bottom topography.

After the client receives the XML string, it must be parsed to extract the useful information. A second class, called MetcastTestMain was written to written to navigate through the entire call-parse-use process. MetcastTestMain instantiates a MetcastClientTest object and sets a series of parameters that is passes to the constructor. These parameters are the same values (latitude, longitude, and size) needed to make the call to the Web service. By calling a method in MetcastClientTest, the SOAP call is made and the XML string is returned to MetcastTestMain. The string is then written to an XML file in a specified directory for static storage. Finally, a Simple API for XML (SAX) parser is used to extract the data from the file and place them into Java variables. SAX is a common event-driven parser that is excellent for XML documents that only require one

67

review. The SAX parser checks each element it comes across against a series of if statements that the programmer defines. The programmer then decides what action is to be taken for each element. In the example below, if the element is defined by "SoundSpeedValues" (as described by the schema), then it will save the string of values to the variable "soundValues."

```
if (element.equals("SoundSpeedValues") && startFlag == true) {

    soundValues = s.toString();
}
```

Figure 21.   Java code excerpt from MetcastTestMain.java (Appendix C) that parses sound speed profile data from Metcast response.

In the Sonar Visualization Workbench Panel, these data variables are stored in a helper API called the EnvironmentalData package, explained in the next section. An important item of note is that this client is based on a known location of the MWS. The client is not capable of searching a UDDI registry and automatically configuring itself to access the appropriate Web service. As MWS evolves into a true Web service and UDDI registries are common on the SIPRNET, a MWS client will need to possess this capability.

2.   EnvironmentalData Package

The EnvironmentalData package is an API written in Java designed to assist MWS clients to parse and access the returned oceanographic data. The API consists of a container class, called EnvironmentalData, which holds a number of related objects. Each type of object represents one particular environmental attribute and is accessible from the EnvironmentalData class. These objects are currently SoundSpeed, SeawaterTemp, Bathymetry, WindSpeed, and SignificantWaveHeight (or Surface).
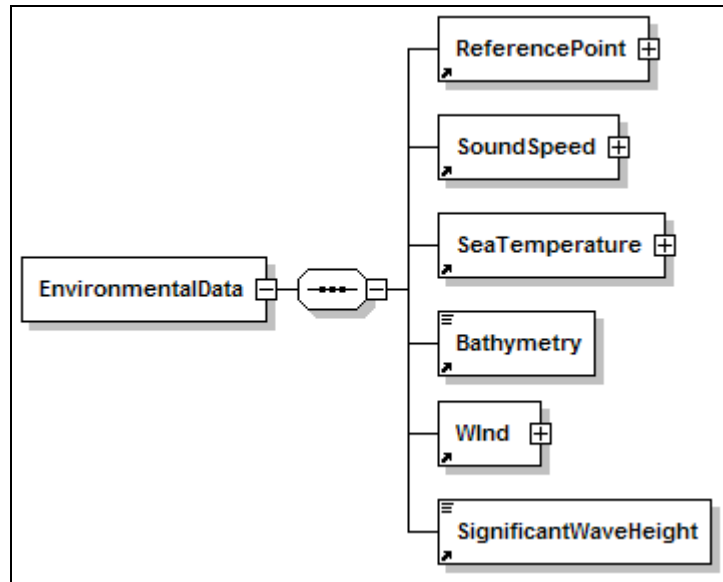
Figure 22.   EnvironmentalData schema element as defined by the Sonar Modeling
          Schema (SMS).

These classes store, format, and translate the data into useful information that can be retrieved directly or through the EnvironmentalData class. Some of the objects, such as SoundSpeed, have the ability to generate charts depicting the data. To allow an application to have access to properly-formatted environmental data, it only has to instantiate an EnvironmentalData object and call the appropriate methods to retrieve it.

C.   SONAR VISUALIZATION WORKBENCH PANEL (SVWP)

1.   Framework

The Sonar Visualization Workbench Panel (SVWP) was originally designed as a standalone product. However, it remained an important requirement within the design process that it can be easily plugged into any parent Java swing application such as the AUV Workbench. The SVWP development process was bound to three integral capabilities. The first requirement was to design an intuitive GUI that guides a user through the sonar modeling and visualization process. Second, the SVWP had to access MWS and be able to feed those inputs locally into an RRA model engine redesigned to accept these inputs. Finally, the SVWP needed a mechanism to display the results. Eventually, the SVWP will be able to access remote model assets when they become

available. For now, the SVWP does possess the capability to view model results that correspond to the proposed schema, but at the time of this thesis only had the capability to use historical files as none of the models were yet exposed on the Web. While Yumetech Inc, a NPS partner, is working on a robust RRA server, the SVWP does have the ability to run a local RRA model that accepts the Metcast inputs. However, the model output does not conform to the master schema. This is because its recursive ray algorithm requires a picking node to efficiently conduct collision detection with pre-positioned receivers. In the meantime, the RRA model results are visualized in a dynamic raypath fashion which possesses a unique and beneficial viewpoint for training operators on the physics of sonar paths. When the RRA server is ready, the dynamic RRA model will still be made available as an option in the SVWP.

2.   GUI Design Evolution

The original RRA code was published in (Holiday, 1998). All of the input parameters were "hard wired" (i.e. statically fixed as constants in the code itself). In 2003, NPS contracted Yumetech Inc. to create a low-level GUI to allow a user to dynamically change sensor inputs and run the RRA model. The generated VRML97 scenes were then displayed in an Xj3D browser window. The result of this contract was the Recursive Ray Acoustic Simulator (RRAS) as shown in the figure below.
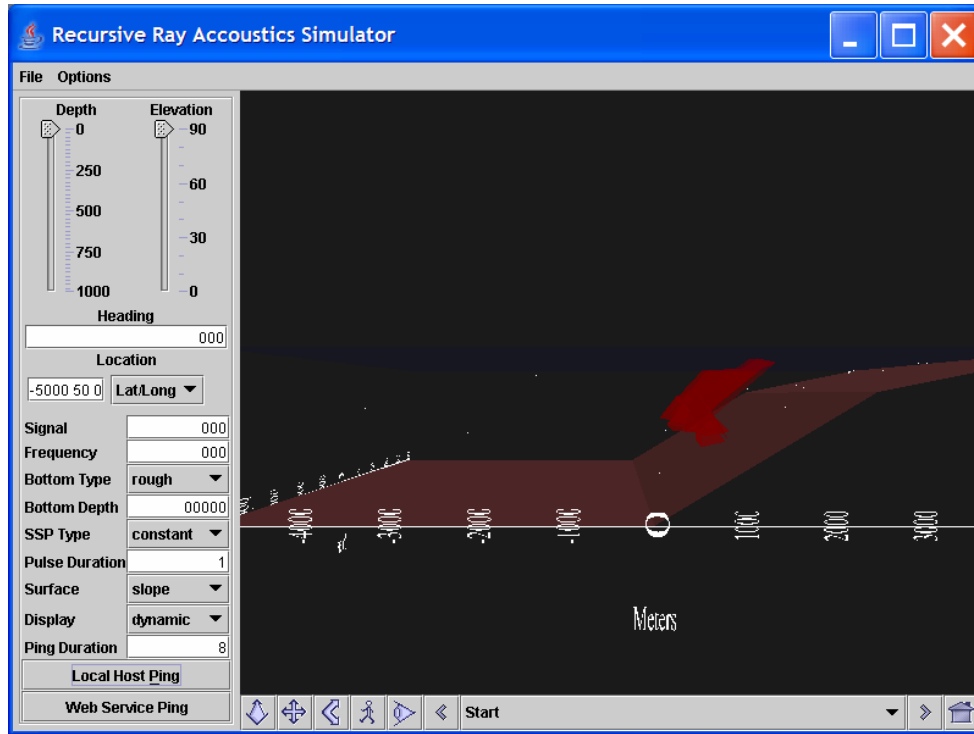
Figure 23. Recursive Ray Acoustics Simulator produces 3D scenes depicting sonar performance in a limited environment.

Note that rudimentary environmental parameters are chosen by the user. The user can select between constant, positive, or negative sound speed profiles; rough or smooth bottom type; and bottom depth. Early work in this thesis first demonstrated how a Web Service architecture might be useful in sonar modeling by incorporating a Web service capability into the RRAS. In the lower left corner of the above figure, a *Local Host Ping* and a *Web Service Ping* are available. The *Local Host Ping* button ran the RRA model as before via a local API invocation. The *Web Service Ping* button recorded the GUI inputs and formulated a SOAP request to a RRA Generator Web service that was created and installed on a local server as part of this effort. Although the results returned from the RRA Generator are no different than those generated locally, it still proved a crucial step in this effort to demonstrate that these model can be ran remotely accessed via Web services with the resultant visualizations returned over the Web. The RRA simply receives the VRML97 representation from the RRA Generator (as a string) and then loads it to the embedded Xj3D browser.

71

The GUI itself as developed by Yumetech was generally straightforward. Along the left side of the panel the user can choose to modify any of the scenario inputs via a range of sliders, drop down menus and text fields. Along the bottom a task bar is present that allows the user to choose which navigation mode to enter and to easily select pre-selected scene viewpoints. After the development of the Metcast Web Service, it became necessary to update the SVWP to integrate this new capability. While the previous GUI only required one section, the new SVWP required a *Metcast* panel, a *Run Model* panel, and a panel to generate static 2D and 3D plots. While at first it seemed the panel might become crowded, it soon became apparent that the functionality inherent in the MWS allowed most of the RRA-related GUI components to be removed. The SVWP with MWS incorporated is shown in the figure below.
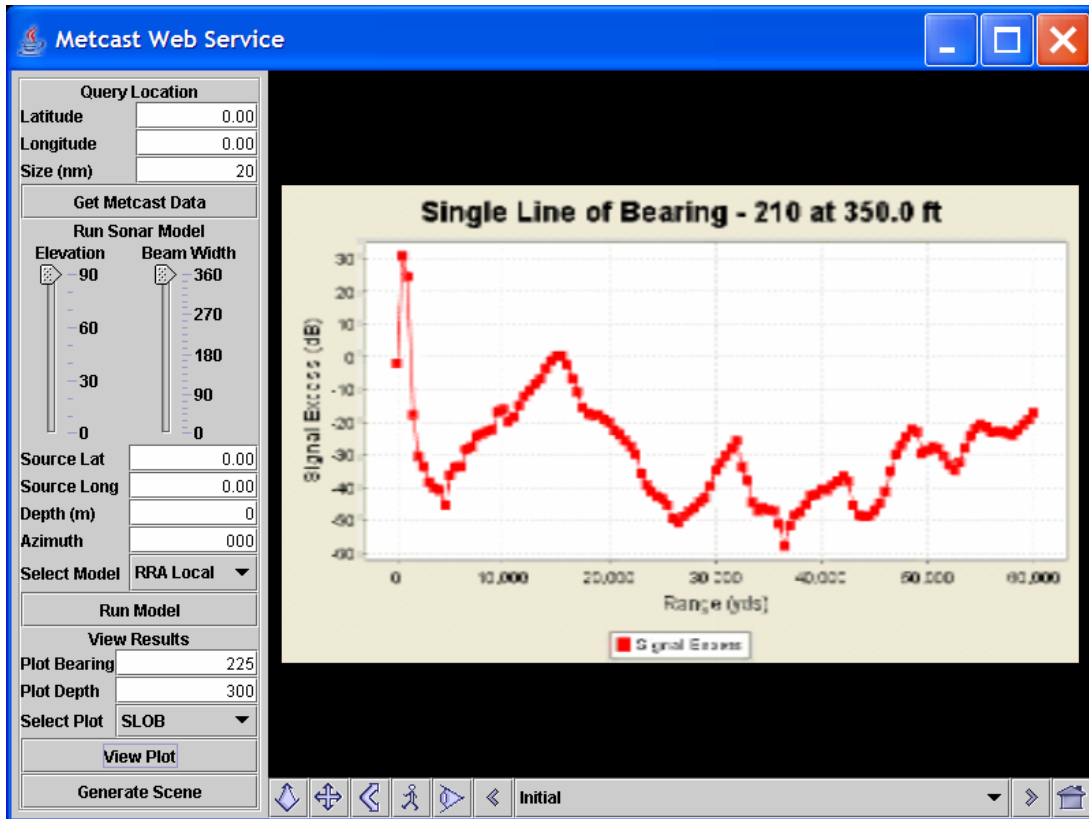
Figure 24.   The Sonar Visualization Workbench Panel GUI implementation autogenerates 2D and 3D graphs such as this single line of bearing plot.

The GUI was designed to provide an intuitive "top-down" approach. The three major sections are the Metcast, Sonar Model, and Viewer panels, typically operated in

order. The Metcast panel allows for the user to enter latitude/longitude position (in decimal form) and the size of the area in nautical miles in the provided text fields. The final element in the panel is a *Get Metcast Data* button that triggers the Web service request to MWS using the entered figures as parameters. The next panel allows the user to set the parameters needed to realize the next step of the modeling process, running the model itself. The *Select Model* drop down box is designed to allow the user to select which sonar model to run. Currently, only the local RRA model is available. In the future, the user will be able to run remote models on RRA and CASS/GRAB servers. After the user set these parameters and clicks on the *Run Model* button, a pop-up menu appears with a list of source templates as shown in Figure 25.
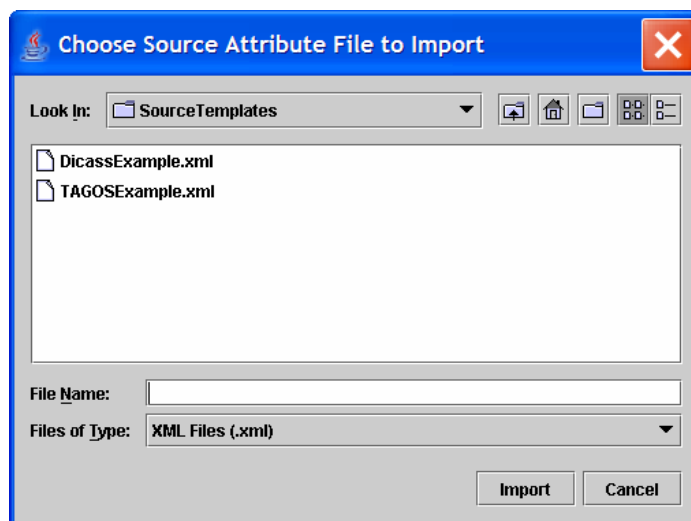
Figure 25.   Source Template pop-up window provides the user with a choice of previously prepared XML files that contain source property information such as frequency of interest.

The pop-up menu allows the user to import a source attribute file. While the user must enter scenario-specific data such as location and deployment depth, the source attribute files contain the source-specific data regarding a particular sensor. For example, if the user wants to simulate a Directional Command Activated Sonobuoy System (DICASS), selecting the DicassExample XML file will automatically import the specific attributes native to that system such as frequency, power, and source level. The templates are validated against the Sonar Source Properties Schema (SSPS). The SSPS, like the

73

MRS, map directly to the SMS and eventually the source template files will validate against a finalized SMS. The attributes section of the SSPS is shown in Figure 26.
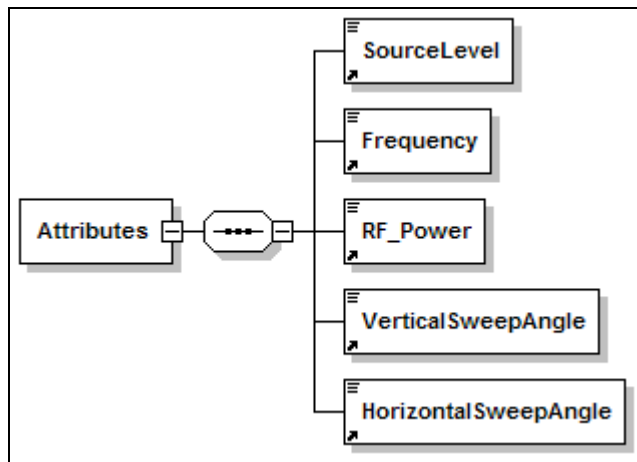
Figure 26.   Sonar Source Properties *Attribute* schema element.

The Viewer panel allows the user to control what visualizations appear in the window. Three types of visualizations are available. The first is the bathymetry scene. If no sonar model has been run, then the *Generate Scene* button updates the Xj3D window with a 3D representation of the bottom topography.
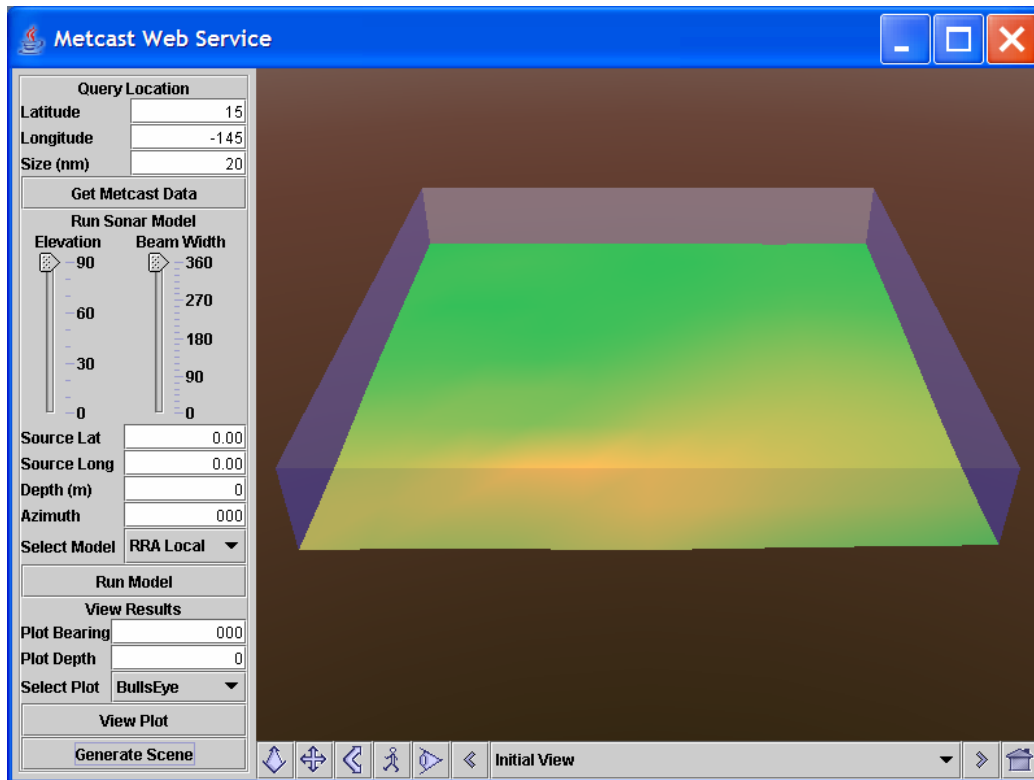
Figure 27.   Once a client receives a response from the Metcast Web Service, environmental data such as bathymetry can be visualized in the SVWP.

The second type of visualization is available if the user runs a local RRA model scenario. The local RRA model, described in detail in the next section, still uses the ray-path code implementation of the RRA model. As a result, activating the *Generate Scene* button updates the Xj3D window with a dynamic 3D scene that visualizes the sonar pulse as it propagates through the water. The pulse color is mapped to transmission loss, allowing the user to see how signal excess might decrease as a sonar beam propagates. While this is a highly useful visualization, the plot viewer may be the most tactically significant. The plot viewer updates the window with one of several 2D or 3D plots.

3.   RRA Engine Modification

The Recursive Ray Acoustics (RRA) model was the first sonar prediction tool to be integrated with MWS in a decision tool application. The RRA code base was originally developed as an implementation of Dr. Ziomek's RRA algorithms as a master's thesis by Timothy Holiday in 1998. The RRA code did not reach full

implementation, however, and was restricted to using simple bottom and sound speed profile exemplar cases as inputs. Instead of bathymetry data, the RRA code allowed the user to choose between a sloped or non-sloped bottom-type at a given depth. Similarly, the user could select from a few generic sound speed profiles. Using such oversimplified parameters, the RRA code was not usable as a general decision tool. The RRA code did lend well as a training aid, as operators could follow a sonar pulse as it propagated through a particular environment. Figure 28 is a screenshot from an exemplar RRA run.
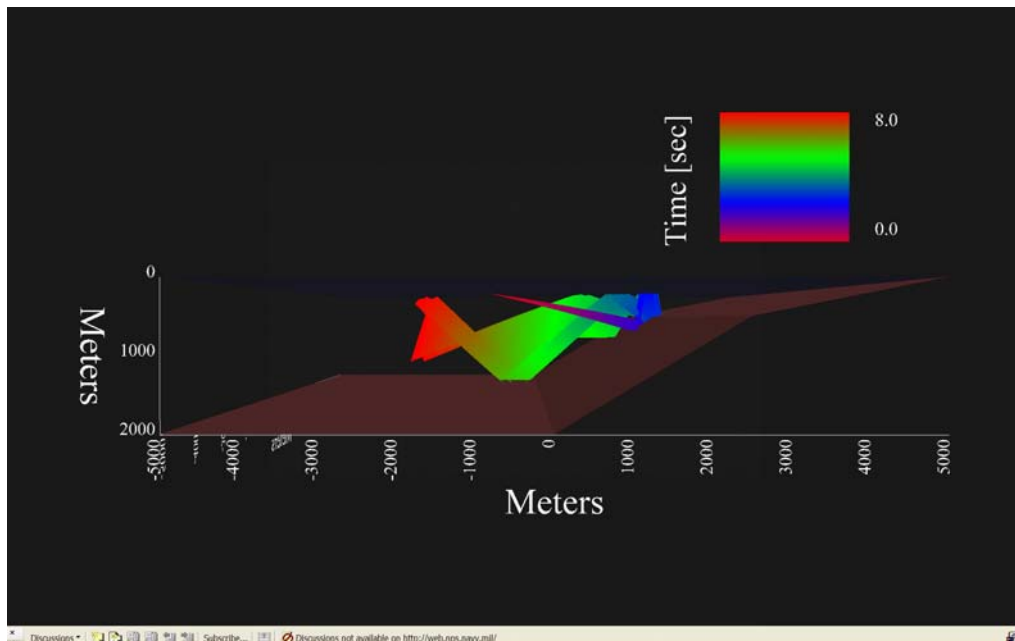
Figure 28.   Screenshot of example RRA visualization (Holliday, 1998).

After a stable version of MWS was exposed on the Web, it became possible to include real-time environmental data in the RRA calculations. While the Sonar Visualization Workbench Panel had the capability of retrieving the data, the RRA code was not mature enough to understand it. Unlike more mature modeling packages, RRA was not yet implemented at a level where it could ingest highly-detailed oceanographic data. Thus, the next step in the SVWP development process was to modify the RRA code to use the real-time Metcast data. It is important to note that the original RRA code never had the capability to ingest real environmental data. A more robust model such as

CASS/GRAB does not require any internal modification; rather it only needs external translation to transform the Metcast XML into the native programming scheme.

The first major modification made to the RRA code was to update the algorithms to use actual sound speed profile (SSP) data. This was a straightforward adjustment made in the SSP class of the legacy RRA code. The SSP class had five prescribed profiles that were selected by a switch dependent on what string was passed into the constructor. These were left in as available options, but a sixth case statement was added that was selected if two arrays were passed into the constructor as opposed to the string. By selecting this case, a new method was called that iterated though the level array until it found the nearest level above and below the current ray depth. Then the sound speed was interpolated based on the distance between those two depths.

The second modification proved to be more difficult. The legacy RRA code simulated interaction with the ocean floor in a simplified manner based on a flat, smooth ocean floor that may be level or slanted. These assumptions greatly simplified the mathematics involved. First, collision detection was able to be determined by checking the ray depth against the bottom depth. Since the bottom depth was either constant or on a constant gradient, this was an easy calculation. Because of the flat surface assumption, the plane normal is trivially calculated and Snell's Law can be applied to determine how a ray reflects off the bottom surface. However, both of these calculations became much more complex upon implementing a full elevation grid as a bottom structure. Unfortunately, a satisfactory solution has not been fully reached as collision detection as some of the more extreme bathymetry examples reveal an interpolation error between calculated ray path and the visualized bathymetry. While a temporary solution is described in the next paragraph, the problem will be solved when the picking nodes are incorporated into the RRA code for a more robust collision detection capability.

The temporary solution to the bottom interpolation problem has several steps. First, a simple check is made to see if further collision detection calculation is necessary by checking the current ray depth against the most shallow bathymetry depth in the area. Thus, the volume of the water column from the minimum bottom depth down can constitute the first bounding box in standard collision detection algorithms. If this passes,

77

then the ray's position in the horizontal plane (x,y) is used to determine the four depths that bound that position. Again, if the current ray depth is above the shallowest of these four depths, then no further collision detection is needed. If this passes, then the bottom depth must be interpolated at the ray position in the horizontal plane. For this calculation, a double-interpolation algorithm was chosen as illustrated by Figure 29.
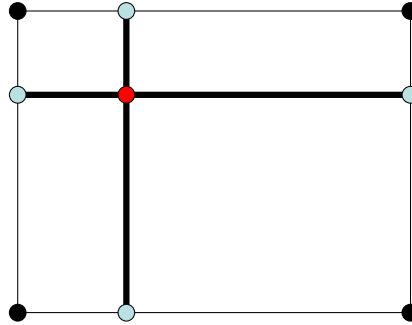
Figure 29.   Diagram of the double-interpolation method.

In the figure, the four solid black circles represent the known depths and the red dot represents the intermediate position whose depth is desired. The first step is to determine the positions of the blue open circles shown in the figure by extending a line along the horizontal axes of the relative coordinate system of the red circle and intersecting it with the outside edges of the box created by the four black solid circles. The interpolated depth of the blue open circles is then calculated from the adjacent corner depths. Then, a second interpolation is conducted between opposing blue open circles. The results in two separate depth interpolations for the current ray position (central intersection depicted by the red circle). In this implementation, a second average taken of the two calculated values is used to determine the final interpolated depth.

Once the mathematics had been worked out, the next task was to update the RRA to properly visualize the new bathymetry. Since the bathymetry data provided from Metcast was packaged as string of points representing a height map, the data translates well to a VRML97 ElevationGrid. The spacing of the grid is converted directly from the resolution returned with the bathymetry data. An assumption made in the RRA code is that the geographic areas are small enough to discount the curvature in the Earth. The

78

resolution of the bathymetric data available on the Metcast Server over the NIPRNET is two geo-spatial seconds for both latitude and longitude, which equates to approximately two nautical miles. After some data manipulation (described in more detail in the Bathymetry Viewer section of Chapter IX), the bathymetry is visualized in 3D and appears to reflect the sonar beams as they collide into the bottom.

Prior to the SVWP, the RRA code examples were designed to follow a single sonar pulse as it propagates through the environment. Advances in common computing power allow a personal computer (PC) or laptop to run more vigorous simulations. If a full 360-degree beam width is selected by the user, the *Run Model* button on the SVWP will trigger a 72 beam sonar simulation. The *Beam Width* slider allows the user to manually adjust the beam angle. Narrowing the beam does increase the resolution within the angle. For example, a 180-degree beam will only simulate 36 beams. The real value of the RRA visualization is not specifically the path of the beam, but the signal strength of the beam as it propagates through the water. This was accomplished by mapping the color of the beams to the calculated transmission loss. This allows an operator to see how the sound is effected as it reacts to its environment as shown in Figure 30.
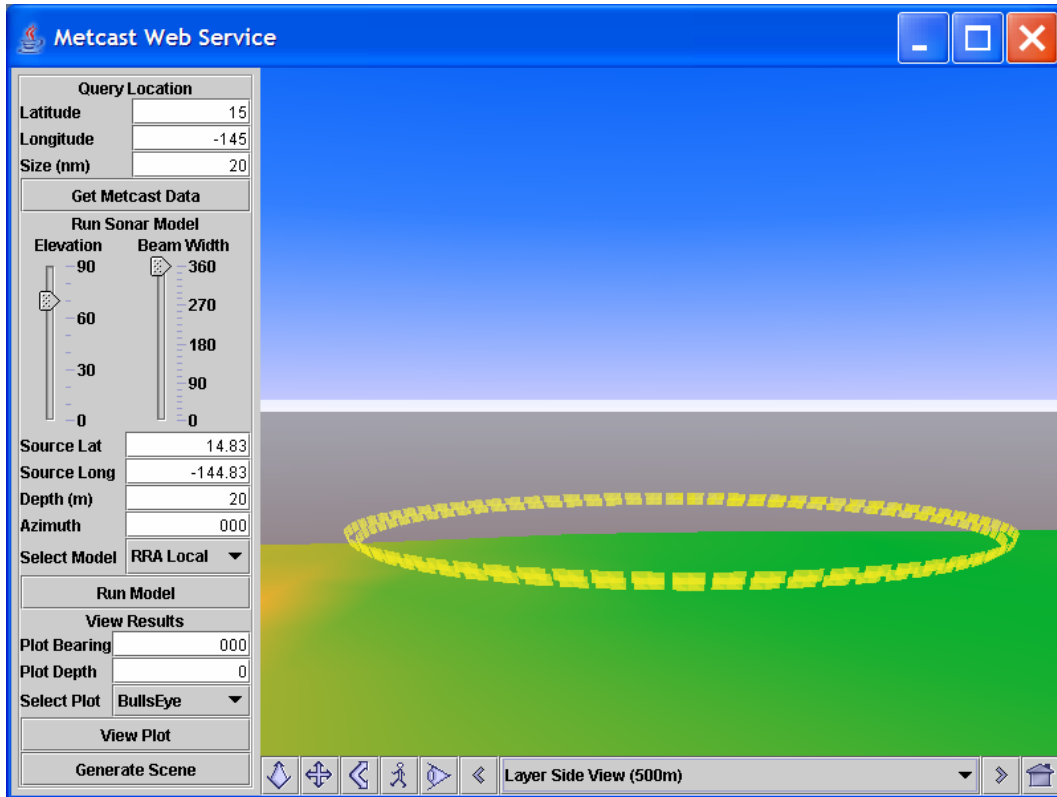
Figure 30. The Sonar Visualization Workbench Panel visualizes 72 sonar beams propagating from a source and uses color to express beam sonar excess values.

4.   View Results Panel

The last panel of the SVWP GUI is the *View Results* section. After running a model, this panel allows the user to select a plot from a drop-down menu and update the Xj3D window. The plots are generated by parsing the model output XML file(s) and calling the appropriate methods in the Sonar Visualization API to create the requested plot. Ideally, the model output XML files would validate against the Sonar Modeling Schema. However, this has not yet been implemented. The only model that has been updated to ingest Metcast data is the CASS/GRAB model through the WEAVER project. Although Sonalysts, the contracted commercial partner in the WEAVER project, has updated the software to output XML, it does not conform to any universal schema.

Having access to sample CASS/GRAB output files, the SVWP was modified to parse these files and create 2D and 3D plots using the Sonar Visualization API. As the CASS/GRAB software is not exposed to external use, the SVWP can not yet remotely run the CASS/GRAB prediction model. However, the SVWP does have the capability to

80

visualize and display them. The visualizations currently available through the Sonar Visualization API are the Bullseye plot, the Stack Scene plot, the Iso-Surface plot, the Sound Speed Profile chart, the Sea Temp vs. Depth chart, and the Single Line of Bearing (SLOB) plot. These are all described in detail in Chapter IX. Figure 31 is an example of a Bullseye plot generated from CASS/GRAB data as displayed in the SVWP.
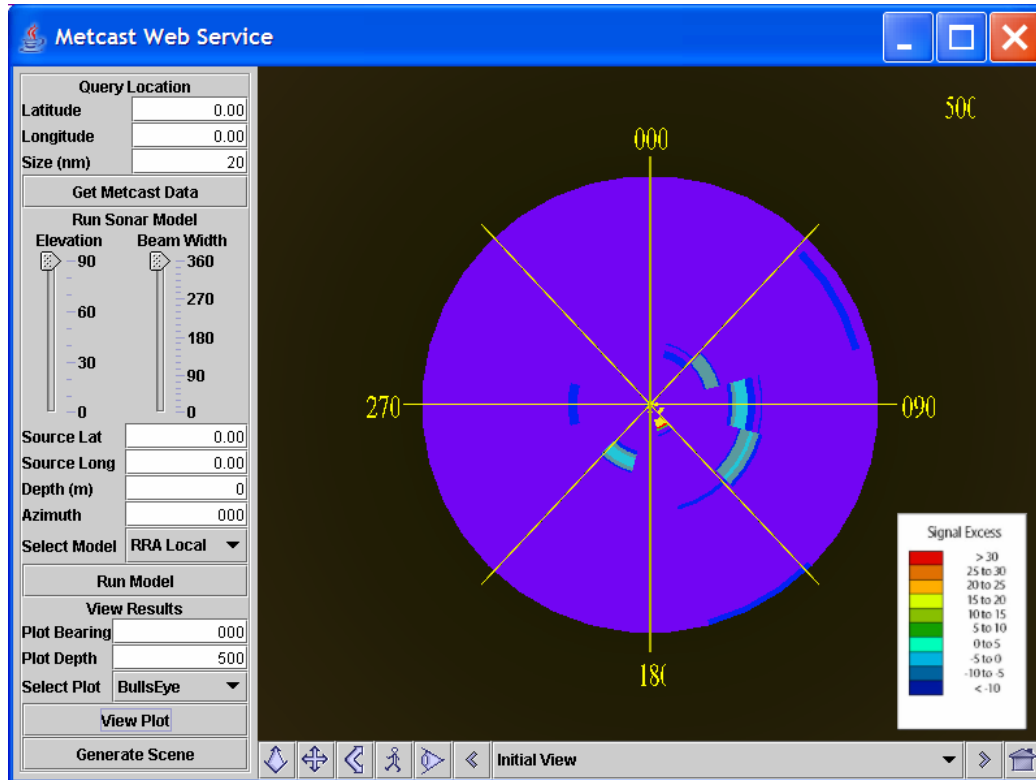
Figure 31.   The Sonar Visualization Workbench Panel can parse XML output returned from an exposed model and generate visual plots such as this 2D Bullseye plot.

5.   AUV Workbench Integration

Integrating the SVWP into the AUV Workbench is an ongoing process that will be continued outside of this thesis. Currently, the SVWP has been installed into the application as a feature, but has not been integrated with the AUV mission planning tools. While it still holds significant value in providing the ability to predict transmission loss characteristics for the AUV sonar in a specific environment, the ultimate goal is to infuse SVWP capability and visualization directly into the AUV panel. With this functionality, a scene may simulate AUV sonar pulses as they move through an environment and use

collision detection algorithms to predict target detection and show real-time coverage areas. More detailed recommendations on future work are found in Chapter X.

Since the SVWP was created as a Java panel, it was easily embedded into the Java-based AUV Workbench GUI. The changes involved instantiating the SVWP classes from an Ant-based AUV Workbench build script instead of a batch file and invoking the panel as a JTabbedPanel in the AUV Workbench's as a child component of a JTabbedPane.

D.   SAVAGE ARCHIVE WEB SERVICE (SAWS)

The Scenario Authoring and Visualization for Advanced Graphical Environments (SAVAGE) Archive contains over 900 X3D models freely available on its web site (http://web.nps.navy.mil/~brutzman/Savage accessed). X3D models are fully importable into any other X3D scene. Thus, once a specific model is created such as an Osa II missile boat, developers can reuse that model as often as desired by importing the model's X3D code into the master scene code.
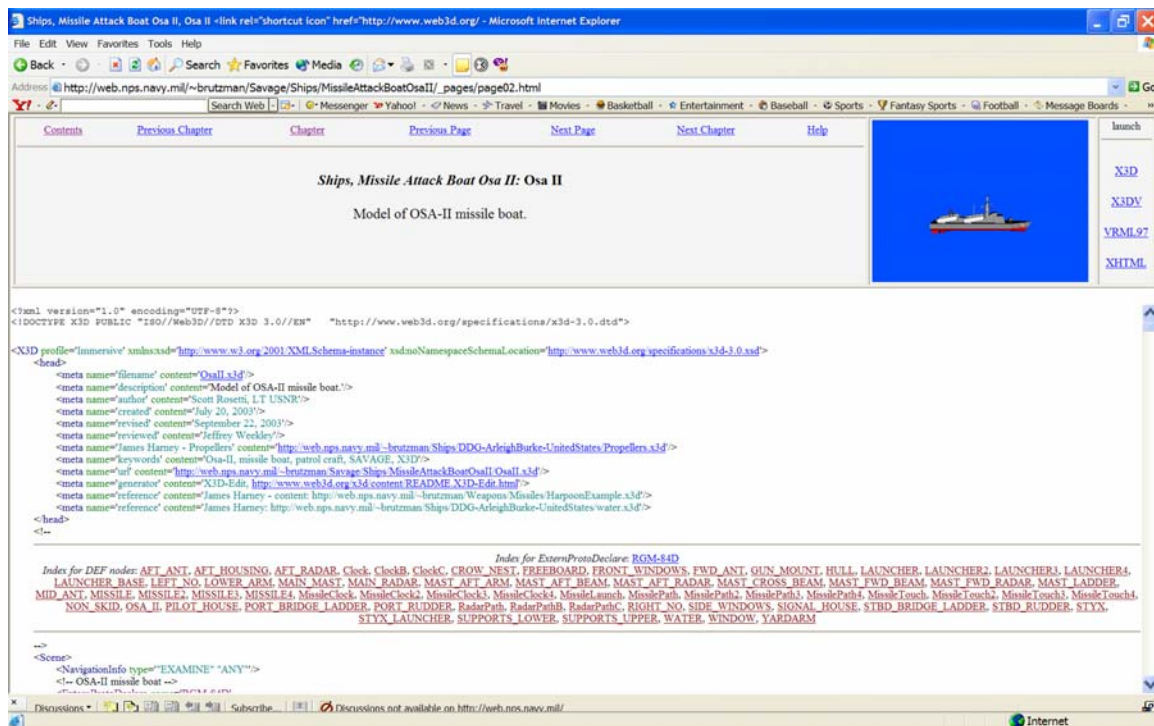
Figure 32.   The Scenario Authoring and Visualization for Advanced Graphical
Environments (SAVAGE) archive allows a user to peruse open-source models by
providing the VRML97 3D scene as well as the X3D code
(http://web.nps.navy.mil/~brutzman/Savage/Ships/MissileAttackBoatOsaII).

As development continues on the AUV Workbench, it is desirable to autogenerate
underwater scenarios in real-time. If an AUV detects a mine, for example, the scene
engine can pull in a model of the newly discovered and dynamically import it into the
scene. An inefficient method of implementation is to create a large, local database of
models that are fielded in the application itself. A better way is to allow the scene engine
to communicate in real-time with the SAVAGE Archive itself. Allowing a two-way
interaction with the SAVAGE Archive ensures that the latest models are used and
removes unnecessary local file storage requirements. In addition, once a scenario plays
out it can be recorded and posted to the SAVAGE archive for future viewing.

The SAVAGE Web Service (SAWS) has been developed to act as the interface
between the SAVAGE Archive and potential users. SAWS exposes three methods to
potential clients: *search, put,* and *receive*. *Search* takes in a list of keywords and searches
the SAVAGE Archive to find potentially suitable models. *Put* accepts a properly

formatted X3D model from the client and posts it within the SAVAGE Archive directory structure. The model then becomes a part of the archive and is freely available to others to download and use. Finally, *receive* received a specific model name and returns an X3D string representation of the model to the client. The client can then import this string directly into a master scene. Future work includes making the service live for full-time Web use.

E. SUMMARY

After implementing the architecture as a web page as part of the WEAVER project, it was desirable to develop a second implementation that demonstrated that an application designed for other purposes can easily be incorporated into the framework. The AUV Workbench, an AUV mission planning tool, was fitted with a panel that accesses MWS, runs a local sonar model, and creates and displays the same visualizations provided by the WEAVER site. The information can then be transformed and used within the mission planning engine.

VIII.   SONAR MODELS

A.   INTRODUCTION

This chapter discusses the sonar models used in the example implementations developed in this research. Currently, only the Recursive Ray Acoustics and Comprehensive Acoustic Simulation System were available during the development of this thesis. In the future, other models may be exposed to the framework and the architecture itself may have to evolve to accommodate them.

B.   RECURSIVE RAY ACOUSTICS (RRA)

The RRA algorithm is a simple, fast, and accurate model that can compute position, angles of propagation, transmission loss, bottom bounce, bottom absorption, travel time, and path length along a ray path in three dimensions. RRA has proven to be a precise model in a wide variety of surface, water-column, and ocean-bottom environmental conditions. RRA was introduced as an extension of ray optics algorithm in (Ziomek, 1993). Using arc length as an independent variable, the RRA algorithm utilizes a series of difference equations to recursively solve the unit vector along a ray path and express it in terms of sound parameters. Due to its capabilities as an accurate yet fast algorithm, RRA was implemented in (Holliday, 1998) as part of a 3D sonar modeling and visualization package.

As solved in (Ziomek, 1995) and summarized in (Holliday, 1998), RRA uses ray theory simplification to solve the linear acoustic wave equation. A brief description of the RRA algorithm suffices here. First, the assumption is made that the acoustic source is a time-harmonic oscillator so the Helmholtz equation can be used as a starting point. Then, by assuming that amplitude varies much more slowly than phase and keeping path length increments small enough to consider the path-step a straight line, a recursive solution can be derived to calculate the position and normal of a wave front as it propagates through the water. Also, the intensity of the wave front can be solved by developing a time-averaged intensity vector that can be integrated over the volume of a ray bundle. These

RRA formulations are both accurate and computationally efficient, and thus, are well suited for real-time sonar modeling.

C.   CASS/GRAB

The Comprehensive Acoustic Simulation System/Gaussian Ray Bundle (CASS/GRAB) model is a U.S. Navy approved acoustic model used to predict the performance of active and passive ocean acoustic systems. Developed in 1993 by the Naval Undersea Warfare Center Division Newport, it has the ability to model monostatic and bistatic active sonar as well as passive sonar in a range-dependent manner. The CASS model is an improved version of the Generic Sonar model and contains a full suite of environmental models and can model most useful sonar settings and parameters across a broad range of frequencies. The model uses the GRAB eigenray model to compute propagation loss.

As the focus of current naval operations has shifted from blue-water deep ocean to brown-water environments, the ability to model the littoral ocean environment has become paramount. One of the advantages of the CASS/GRAB model is that it performs relatively well modeling high-frequency sonar systems that are commonly used in littoral environments. Many range-dependent acoustic models are not designed for high-frequencies and become computationally intensive above frequencies of several kilohertz. The GRAB model's main function, however, is to calculate eigenrays in range-dependent environments in the frequency band 600 Hz to 100 KHz (Chu and others, 2002). This capability makes CASS/GRAB a highly effective tool for modeling high-frequency systems in the littoral environment and has led to it becoming the Navy-standard sonar performance model for frequencies above 600 Hz.

CASS uses the GRAB eigenray model to simulate sound propagation loss as a function of range and depth. Input parameters to CASS are entered via simple formatted external text files. GRAB sorts generated eigenrays into families of comparable numbers of turning points and boundary interactions (Chu and others, 2004). The properties for these rays are then power averaged for each family to produce one representative eigenray. Signal excess is then computed by summing all the eigenray path combinations

86

and finding the peak signal-to-noise level. The main difference between GRAB and classical ray theory is that the amplitude of the Gaussian ray bundles is global rather than local, thus affecting all depths. The weighted averages of the ray bundle properties are then used to calculate propagation loss (Chu and others, 2004).

D. SUMMARY

Ultimately, it is desirable that all sonar models be exposed on the web and conform to common standards to promote interoperability. RRA and CASS/GRAB are the first sonar models to be "wrapped" in code that acts as an interface between native parameters and XML-based messaging. By creating these wrappers, disparate models can become accessible to any client that also conforms to the framework. A great deal of productive future work is expected.

THIS PAGE INTENTIONALLY LEFT BLANK

IX. SONAR VISUALIZATION DESIGN

A.   INTRODUCTION

This chapter discusses the sonar visualization API and the development of individual sonar plots. The Sonar-Visualization API was developed to parse conforming XML files and autogenerate various 2D and 3D plots that effectively and intuitively provide tactically relevant information regarding the undersea operational environment.

B.   SONAR VISUALIZATION

Visualization design is a vital component of any decision aid's development process. Without the ability to effectively convey the information at hand, a decision tool is useless. It is thought that all human reasoning may be constrained and guided by external aids (Norman, 1993). Without these external aids, the brain has difficulty synthesizing diverse data into information, and then information into insight. Fortunately, the human mind is adept at interpreting and modifying these aids to enhance its own cognitive ability. It is particularly adept at understanding graphical representations of data that use visual cues such as color or shape to signify a more complex meaning. Navigational charts, weather maps, and architectural blueprints are some common visual data representations. Even using a pencil and paper to perform long division is a form of visual cognitive aiding (Card and others, 1999).

According to (Card and others, 1999), visualization is defined as the use of computer-supported, interactive, visual representations of data to amplify cognition. Since it is true that computers cannot possess situation awareness in its truest sense, it is also true that computers cannot display or convey it directly to the user. However, computers can express information in an intuitive way that fosters cognition and insight. Well-constructed real-time data visualizations can cultivate a deep and rapid grasp of the tactical environment. The improved situational awareness can transfer into sharper operator decision-making and superior overall operational performance.

It is true that computers have changed the face of data representation. Through its ability to handle large sets of scientific data and render some graphic abstraction of that

data, computation power has become a remarkable cognitive aid. However, achieving success is not always simple. Some data sets are difficult to convey in an intuitive way. In general, data that represents physical elements are capably visualized in a 3D format. This is somewhat true for sonar visualization. Bathymetry, ships, and even the sonar beam itself can be effectively modeled in 3D in a realistic manner. However, in order to be fully aware the undersea picture, the warfighter needs to understand the relationship and interactions between the environment and sonar propagation. Abstract parameters such as sound speed, temperature of the water, and the transmission loss of a sonar beam as it moves through the water are more difficult to simultaneously visualize into an integrated, spatial 3D picture.

Although 2D and time-varying sonar visualizations are well understood, dynamic 3D sonar visualization is a new field and not well developed. Many sonar modeling and visualization tools have been created over the years to provide insightful representations of the underwater picture. An additional advantage to these legacy visualization formats is that, over the years, operators have become accustomed to them and already possess an internal algorithm to convert these pictures into valuable situational knowledge. Some of the visualizations in this thesis build off these standard plots while some attempt to create innovative representations. All of them seek to externally promote the cognition of the user to attain situational awareness and improve tactical decision making. This area of research deserves much further work.

C. VISUALIZING ENVIRONMENTAL DATA

1. Bathymetry Viewer

One of the shortcomings of the visualizations generated by the legacy RRA code is the lack of real or historical bathymetry data. In the RRA code, the ocean bottom was approximated as a smooth plane that could be given a constant gradient. In addition, the surface was also simulated by a smooth plane. While these RRA visualizations still maintained value in that an operator could see how sonar is affected by its environment, the software tool lost much of its accuracy due to these gross approximations. Figure 33 shows a single sonar ray as it reacts with the bottom and surface.
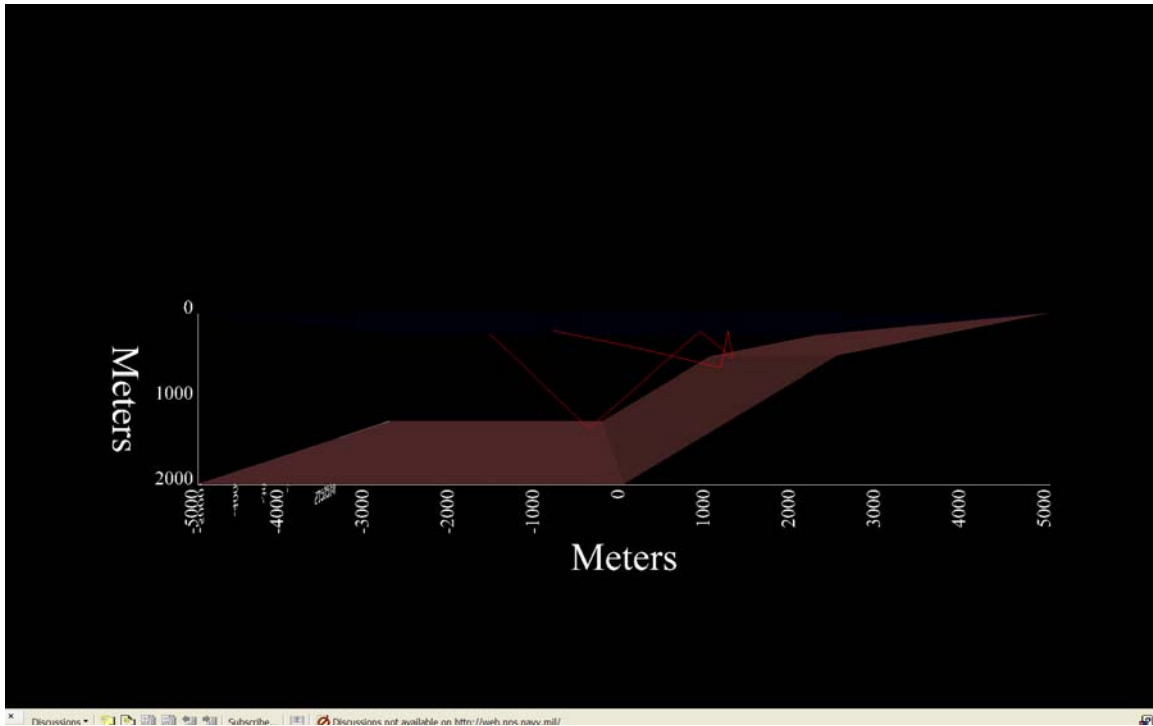
Figure 33.   The Recursive Ray Acoustic code base autogenerates 3D scenes such as this
raypath as viewed in Cortona.

Once world-wide bathymetry data was made available to this project through the
development of the Metcast Web Service described in Chapter VI, the Sonar
Visualization Project and RRA code-base possessed the means to incorporate real-time
bathymetry and oceanographic data into its modeling package. This greatly enhanced the
accuracy of the RRA model output as well as improved the visualizations by providing
detailed bottom characteristics. The bathymetry data is extracted from the Metcast
response XML stream and captured in a Java object (appropriately called "Bathymetry).
Since the bathymetry data provided from Metcast was packaged as a string of points
representing a height map, the data translates well into an X3D ElevationGrid. An
ElevationGrid is an X3D construct that is often used to visualize terrain or complex
structures. An ElevationGrid node requires a grid of heights (or in this case depths), the
number of columns, the number of rows, and the spacing between each point.

Assuming the positive X axis of a VRML virtual world points north, the
ElevationGrid is formed by depths in a string from west to east, then north to south.
Unfortunately, the actual data string returned from Metcast plots depths from east to

91

west, south to north. In addition, RRA specifies depth as a positive value, while in Metcast; a positive depth is above sea level. Thus, the string had to be tokenized and instantiated as an array. It can then be manipulated before it was retranslated back into a string to be used imported as an X3D field. Figure 34 shows an exemplar ElevationGrid node with some of the data excerpted in the fields themselves.

```
<ElevationGrid creaseAngle='1.57' height='0.0 0.0 …. 0.0 0.0 ' solid='false' xDimension='6' xSpacing='4000' zDimension='6' zSpacing='4000'>
        <Color color='0.0 0.6875 ….. 0.6875 0.19 '/>
</ElevationGrid>
```

Figure 34.   ElevationGrid node from Bathymetry scene (note: some data elided from "height" and "color" fields to conserve space).

In this example, the defining nodes are xDimension, zDimension, xSpacing, zSpacing, height, and color. The dimension fields specify the number of heights in the x-direction while the spacing fields indicate the number of meters between the heights. Thus, each row in this example has six heights spaced at 4000 meters apart. The spacing is directly determined from the resolution of the Metcast data. On the NIPRNET version, the resolution is two degree-seconds, or approximately two nautical miles (4000 meters). Thus, simple arithmetic tells us that the length of this example is equal to 20,000 meters or ten nautical miles.

The height field contains the values correspond to depths at each geographic position. Each height is given a color value based on depth. The shallowest depth is green and the deepest is a light brown. The colors of all the depths in between are interpolated between these color values based on depth. When visualizing the bathymetry data, it is difficult to assess true depth from just the ElevationGrid. In order to fully convey the environment in 3D, the surface and water column also need to be rendered in 3D. This was accomplished by rendering a semi-transparent water column that extended from sea level to the bottom floor. This required implementing methods in the bathymetry class that return the depths of each side of the bottom floor. With this data, geometry can be drawn to represent the water column boundaries. Figure 35 shows an exemplar bathymetry model of the southern shore of the Hawaiian island Kauai.
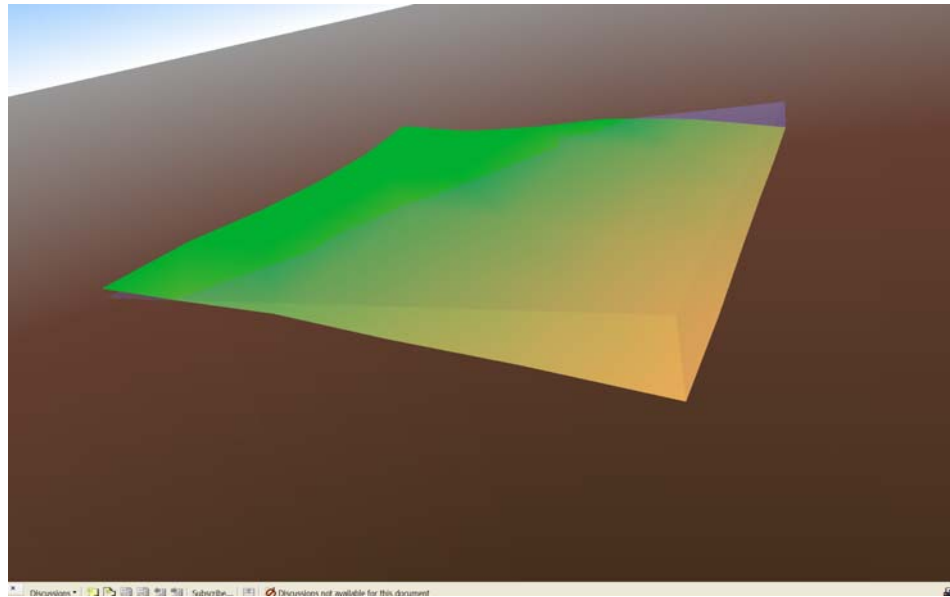
Figure 35.   Bathymetry is render using an ElevationGrid node, as seen here in a
VRML97 scene as viewed in Cortona. No vertical exaggeration was added.

In addition to the bathymetry, it was evident that many of the other fields returned from Metcast needed to be visualized to help convey their relationship to sonar performance. While many of these parameters can be focused on in separate visualizations, they can only clutter the bathymetry scene when viewed in combination. Thus, it was determined that data such as sea temperatures, sound speeds, wind speed, and significant wave height need to be displayed to the user via a heads-up display (HUD). The EnvironmentalData API provides methods to create image files of sound speed profile and sea temperature versus depth charts as well as manipulate, format, and display weather and location data. Figure 36 is a demonstration of the Metcast visualization with HUD.
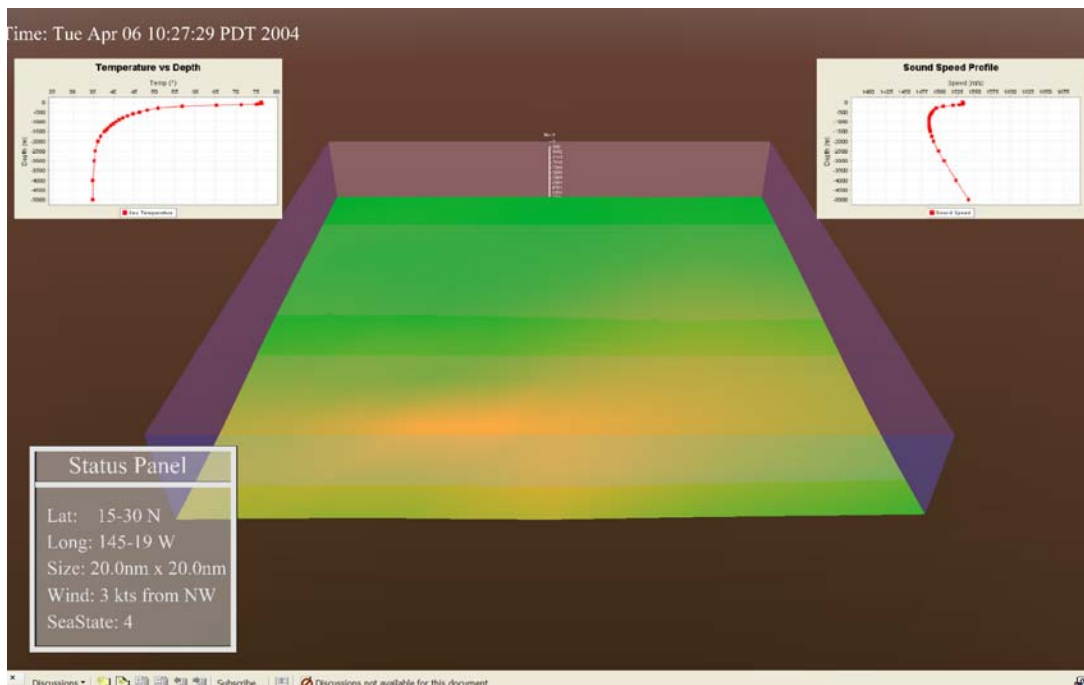
Figure 36.   When a heads-up display is added, addition valuable tactical information can be seamlessly provided as in this scene.

2.   Sound Speed Profile (SSP)

The sound speed profile (SSP) is a visualization construct common throughout undersea warfare. It is generally depicted as a 2D graph showing the relationship between depth and the speed of sound in water at a specific geographic point in the water. The shape of the profile can tell an operator a great deal about how sonar will propagate as it passes through this location. The path sound will take in the ocean depends upon the velocity gradient through which it is traveling. This gradient can be positive, negative, or constant depending on how temperature varies with depth.

Because the ocean is not homogeneous, sound speed can vary greatly from the surface to the bottom. Typically, a SSP will contain more than one, if not all, of the gradient types. Sound rays will react differently depending on the gradient of the water. A positive gradient causes a sonar ray to bend towards the depth of minimum sound speed. Sonar rays that travel within a constant SSP gradient continue to travel in a straight line. This is also true for a slightly negative temperature gradient, as the pressure effect offsets

94

the temperature effect. A negative sound speed gradient bends rays downwards toward the depth of lower sound speed.

The JFreeChart API was used to generate the sound speed profile charts used in this thesis (http:www.jfree.org/jfreechart). The sound speed data returned within the full set of Metcast data is parsed and instantiated as two double arrays, one for depth levels and one for the sound speed values. The levels and values were matched by the array indices. These arrays are provided as inputs to the x-and-y axis values of a JFreeChart XYPlot. Figure 37 shows an exemplar sound speed profile chart. The chart can be exported and saved as a JPG or PNG file.
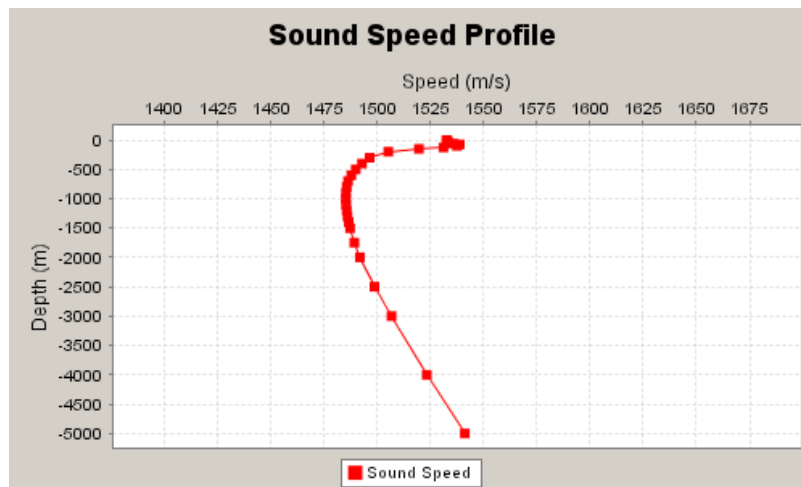
Figure 37.   Example Sound Speed Profile plot produced using JFreeChart
(http://www.jfree.org/jfreechart).

3.   Sea Temperature vs. Depth

Another chart useful in undersea warfare is one that shows the relationship between depth and temperature at a specific geographic location. Commonly known as a bathythermograph (BT) plot, it represents the observable temperature data that are the primary input values in calculating of the sound speed profile (SSP). The BT plot is useful in that it provides a graphical representation of the thermal structure of the water. Using the BT plot, the water column can be separated into a three-layered structure. These layers are the Mixed Layer, Main Thermocline, and Deep Layer. Sound reacts differently as it passes through each layer. Much of what can be gleaned from a BT plot

is also represented in the SSP plot. Although the sound speed profiles are more common, many operators prefer to use the BT plot in its place. Any versatile decision aid ought to be capable of displaying both plots depending on user preference. Figure 38 shows an exemplar sea temperature plot rendered using the Java-based JFreeChart API (http://www.jfree.org/jfreechart).
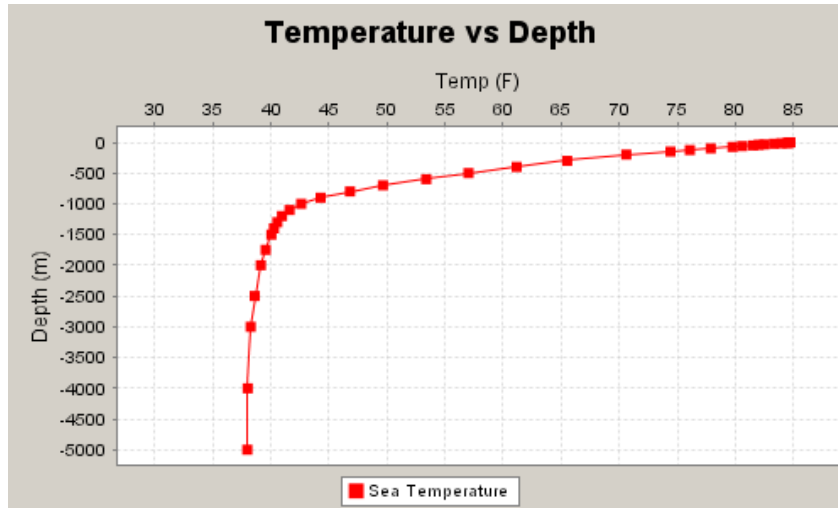
Figure 38.   Example Sea Temperature vs. Depth plot produced using JFreeChart (http://www.jfree.org/jfreechart).

D.   SONAR-VISUALIZATION API

1.   Model Output

While the XML schema proposed in this thesis is a good starting point towards implementing a standard for USW modeling and simulation, it is certainly not yet thorough. As part of the WEAVER project, Sonalysts has developed its own XML schema to govern the output from a CASS/GRAB model. In order to assist in the development process, Sonalysts developed its initial schema to translate directly from traditional CASS/GRAB output files with the intention of eventually developing a more general schema. While this assists in troubleshooting the CASS/GRAB software, it does not lend well to universal compatibility. In order to conduct the visualizations demonstrated in this thesis, either the sonar visualization API needs to parse directly from the Sonalysts schema, or else transform data from the Sonalysts schema to the Sonar Modeling Schema and then parse. After reviewing the schemas, it was determined that

96

although the internal organization differed, the data structures were similar. Thus, it judged more efficient to parse the Sonalysts data directly and eventually make relatively minor changes to the parser once a master schema is adopted by Sonalysts and NPS together.

With that in mind, the CASS/GRAB schema described sonar excess data for a particular depth. Thus, one XML file corresponded to one depth level. Eventually, these files ought to be integrated into one file. In each file, sonar excess ratios are given organized per specified range and radial. The example output files that will be demonstrated in this thesis contain twelve radials with 121 ranges per radial. There were ten files, each corresponding to separate depth. No positional information was available in these files. The Model Output portion of the SMS incorporates the separate files into one master file and the SMS would also include model input and environmental data as well to enhance the visualizations. Currently, the environmental data is imported separately.

2.   SonarModelOutputParser

The SonarModelOutputParser is a Java class in the Sonar-Visualization API that uses the SAX parser to import sonar modeling output data in XML form. Currently, the parser is designed to understand XML generated from Sonalysts CASS/GRAB schema, but can be eventually updated to parse more general files. The SAX parser reads the relevant parameters from the file and instantiates them as Java objects or variables. The data is then used to form two separate constructs. When visualizing 3D volumetric data, it is often beneficial to display cross-sections of the data so they user can see trends within the volume. This is especially vital in sonar visualization as variable conditions can cause shadow-zones close to the source and still have strong sonar levels at greater distances. With this in mind, the SonarModelOutputParser divides the data into two constructs called RadialBlades and DepthBlades. RadialBlades contain sonar data relative to depth and range along a specified radial. These may be compared to thin slices of cake, allowing the user to see the layers within the cake. DepthBlades describe sonar data relative to radial and range within a specified depth. DepthBlades appear as rings, or disks, of data and represent vertical slices of the volume.

Using these two constructs, many useful displays can be created in both two and three dimensions. The SonarModelOutputParser contains methods to create a number of sonar visualization plots in X3D by utilizing other classes within the Sonar Visualization API. These X3D plots can then be viewed in supported web browsers including Microsoft's Internet Explorer or Xj3D. Currently, the Sonar-Visualization API has classes to generate common plots such as the bullseye, single line of bearing, and full field plots. It also generates a few innovative plots such as the Stack Scene plot and the Iso-Surface plot.

3. Single Line of Bearing (LOB) Plot

The single line of bearing (LOB) plot is straightforward but useful. It is a 2D plot that displays signal excess over range for a given bearing and depth. Signal excess is a very valuable piece of information as it is defined as the amount (in dB) of signal in excess of that required for an operator to detect 50% of the time. So, the greater the signal excess in an area, the greater the chance of detecting a target. As signal excess varies greatly in relation to depth, range, and direction; it is often difficult to visualize in 3D. In order to clearly convey how the values vary, it is often beneficial to hold two of the variables constant. While this provides a more concise plot, doing so narrows the focus of the visualization to a single bearing, depth or range. Nevertheless, they are common and valuable plots. One of these plots is known as a single line of bearing (SLOB) plot. In a SLOB plot, the depth and bearing is held constant. The result is a snapshot of the signal excess values as a sonar pulse travels from the source.

By examining Figure 39 below, an operator can gain valuable information about how sonar is propagating along the user-selected bearing of 210. As might be expected, a strong signal excess is evident for two-three thousand yards due to direct path propagation then it decreases rapidly. Around 15-20 thousand yards, signal excess increases again at this depth probably due to convergence zone or bottom bounce propagation. It also appears similar peaks are present around 32,000 and 48,000 yards from the source. This information can be quite helpful in USW mission planning.
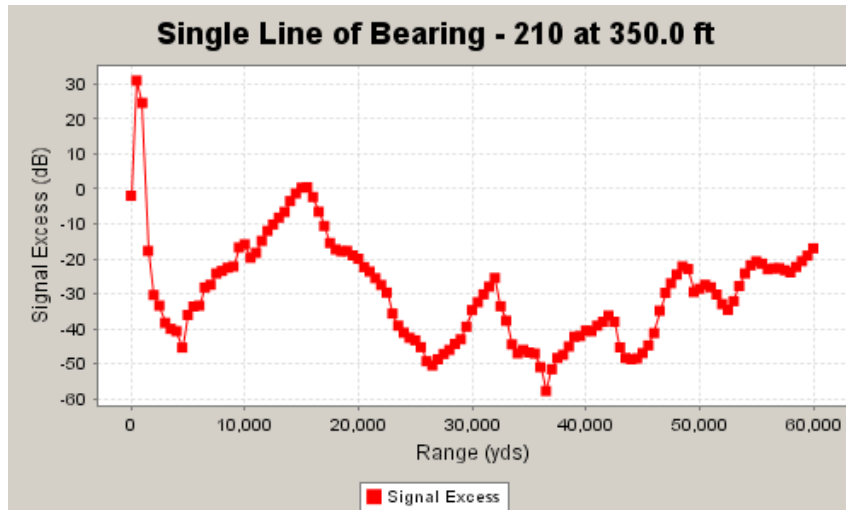
Figure 39.   Single Line of Bearing plot showing signal excess values vs. range as produced using JFreeChart (http://www.jfree.org/jfreechart).

4.   Bullseye Plot

Another common plot in USW is the bullseye plot. The bullseye plot displays signal excess values for a user-selected depth. The signal excess values are color coded in order to allow a user to quickly determine the spatial areas where signal excess is high and thus probability of detection is greater. The legend identifies the range of signal excess values that each color represents. In developing a color scheme, the "ROYGBP" (Red-Orange-Yellow-Green-Blue-Purple) color-scheme was chosen even though it is generally not considered an intuitive design. However, there are two advantages in this use-case. First, the "ROYGBP" color-scheme is common in USW decision aids and thus fleet operators are accustomed to it. Second, in this scheme, red stands for the high signal-strength levels and is easily discernable from the blue and purple that represents the lower signal-strength levels. In a bullseye plot, the source is located in the center of the rings. Range rings are placed on top of the plot to indicate every 10,000 yards from the source. Bearing markers are also given every 45 degrees to assist the user in discerning direction. Figure 40 shows an example bullseye plot as generated from example CASS/GRAB model output data.
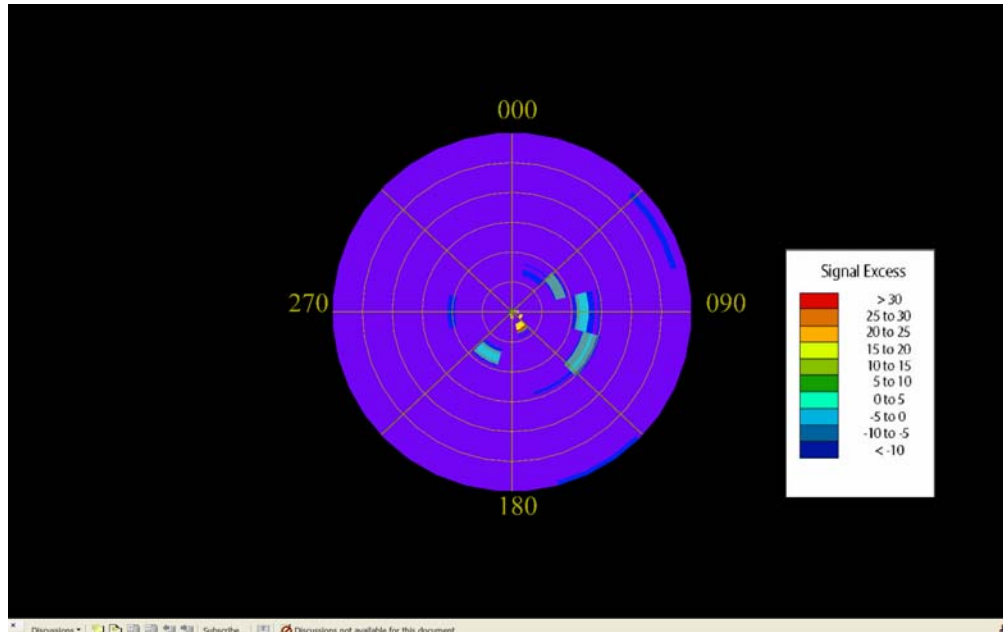
Figure 40.   Screenshot of 3D Bullseye plot in Internet Explorer. The legend on the right
describes the color scheme used in the bullseye plot

The bullseye plot is generated by the BullsEyePlot class in the Sonar
Visualization API. The SonarModelOutputParser instantiates the BullsEyePlot class by
passing in a DepthBlade object. The DepthBlade object provides all of the signal excess
data relative to range and bearing. The BullsEyePlot class then formulates a well-formed
X3D scene. The plot itself is created by creating an IndexedFaceSet for each radial. In
this example, there are 12 radials and 121 ranges. Thus, each data point has a thirty
degree resolution. The discrete ranges for each value point are specified in the
CASS/GRAB model output file. In this example, the greatest range is approximately
60,000 yards from the source. As the 12 IndexedFaceSets are created, each of the 121
data points (one for each range) of each radial are tested in a switch statement to
determine the color to be rendered.

5.   Stack Scene Plot

The stack scene plots were developed as part of the WEAVER project in an effort
to deal with the occlusion problems in volume visualization. The volume occlusion
visualization problem refers to the fact that it is difficult displaying non-homogeneous 3D
volumes. Typically, these volumes are displayed in one of two ways. The first way is to

100

display a representative 2D cross-section. These representations can be meaningful but can leave out valuable information. The second way is to visualize the entire 3D volume. This provides a better overall picture, but also may make critical interior data points difficult to see. The stack scene plot attempts to merge the two in what could be called two-and-a-half dimensions. This plot takes several 2D bullseye plots and "stacks" them as they would appear in 3D. The bullseye plots are thus "pulled apart." That is, the distance between the plots can be exaggerated in order to allow the user to see in between the plots to get a feel how the plots change relative to depth. An interactive HUD allows the user to activate and de-activate each individual bullseye plot in order to "get a better look." Figure 41 shows an example stack scene plot.
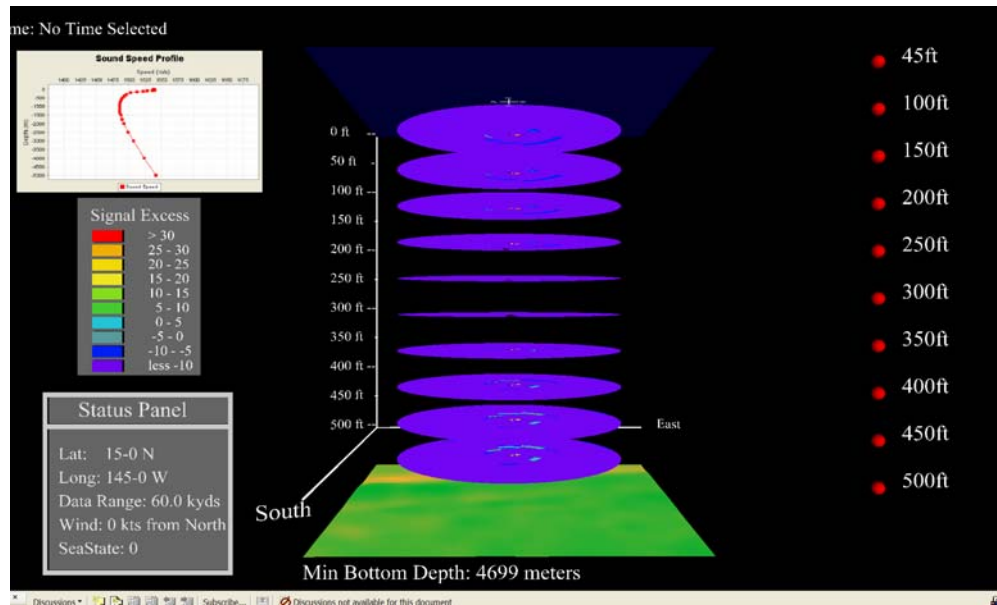
Figure 41.   The 3D Stack Scene plot (as viewed in Internet Explorer) shows multiple signal excess cross-sections which the user can toggle on or off using the buttons seen to the right of the stack.

The stack scene plots are created by the Sonar Visualization API class StackScene3D. This class is instantiated by the SonarModelOutputParser with a Java LinkedList of DepthBlades. In order to prevent the scene from getting too large, StackScene3D only displays the first ten DepthBlades in the list. Each DepthBlade is then used to create a bullseye plot. These plots are then stacked at their respective depths. In

101

addition, the StackScene3D class contains a method to pass in an EnvironmentalData object which contains the oceanographic data returned from a Metcast query. If this method is called, then StackScene3D can incorporate this data into the scene displayed via a HUD. The HUD contains a status panel, a sound speed profile chart, and two interactive control panels. The status panel displays basic data such as latitude, longitude, wind-speed and sea state. The control panel on the right allows the user to select or deselect bullseye plots at the respective depths. The color panel on the left currently acts as a legend but is rendered as geometry. This was done so touch sensors may be added to the geometry in future implementations to toggle specific colors on and off in the bullseye plots. However, this is not yet implemented. Finally, bathymetry data is used to render bottom topography. Figure 42 shows the stack scene as viewed from above the water.
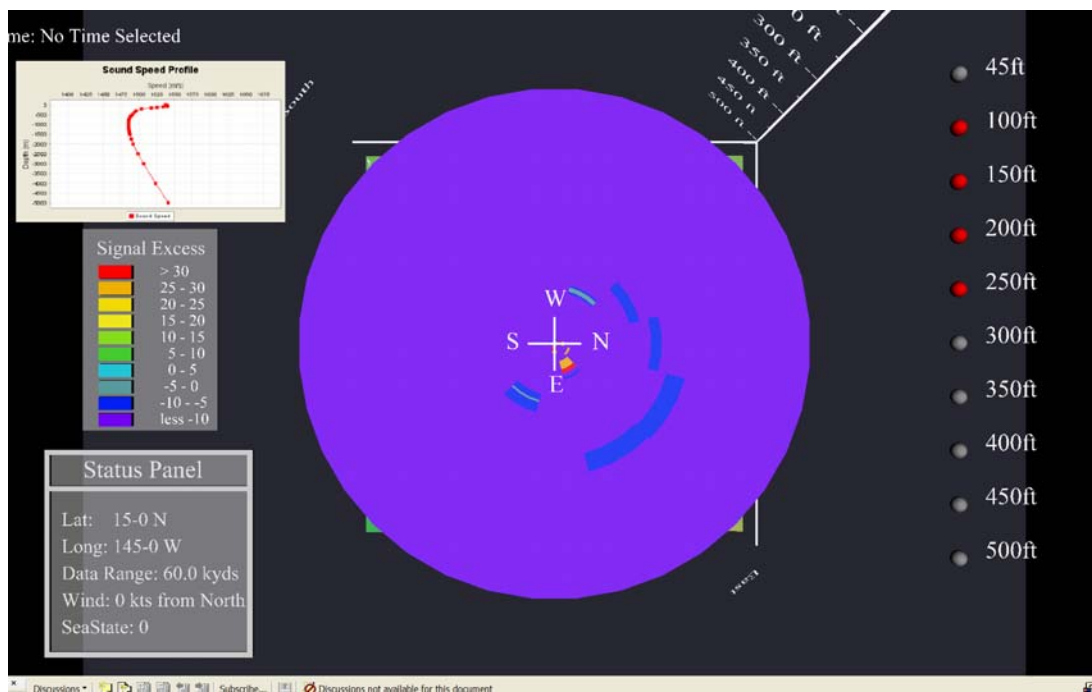
Figure 42.   The 3D Stack Scene plot provides built in viewpoints to allow the user to view specific cross-sections as demonstrated in this scene in Internet Explorer.

Aside from its design, there are two inherent advantages to the stack scene plot. The first is that it is completely interactive and navigable by the user. The user can maneuver freely around the scene to view the scene from new vantage points, and also

choose predetermined viewpoints incorporated into the scene. The user can activate or hide individual bullseye plots by clicking on the HUD GUI embedded in the native scene. Figure 43 is a screenshot taken while navigating the scene. The second advantage is that, since it is X3D, it is viewable in supported web browsers. Thus, it is easily posted to a web site to share with other units.
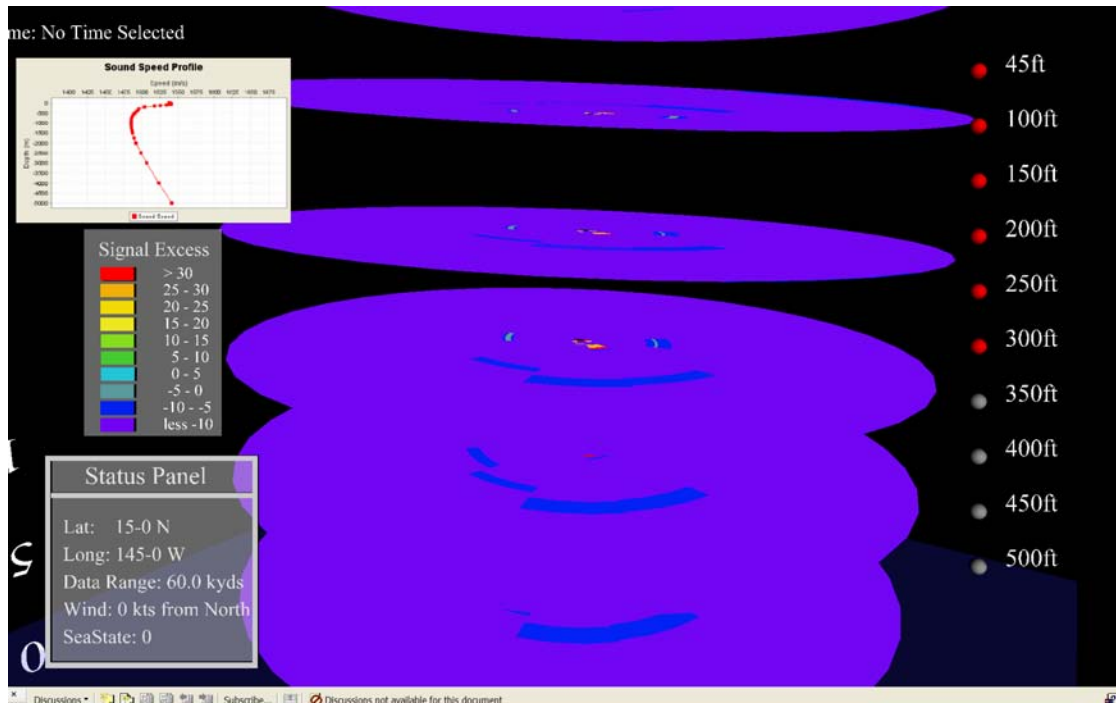
Figure 43.   The 3D Stack Scene plot allows the user to freely navigate the scene to get unique views of the represented data.

6.   Full Field Plot

The full field plot is another X3D implementation of a common analytical plot in undersea warfare. The full field plot displays signal excess as function of range and depth for a specified bearing. As seen in Figure 44 below, the full field plot allows the viewer to understand how signal excess varies as it moves through the environment. Convergence zones, bottom bounce, and direct path trajectories can be easily discerned with this plot. The signal excess values are color coded.
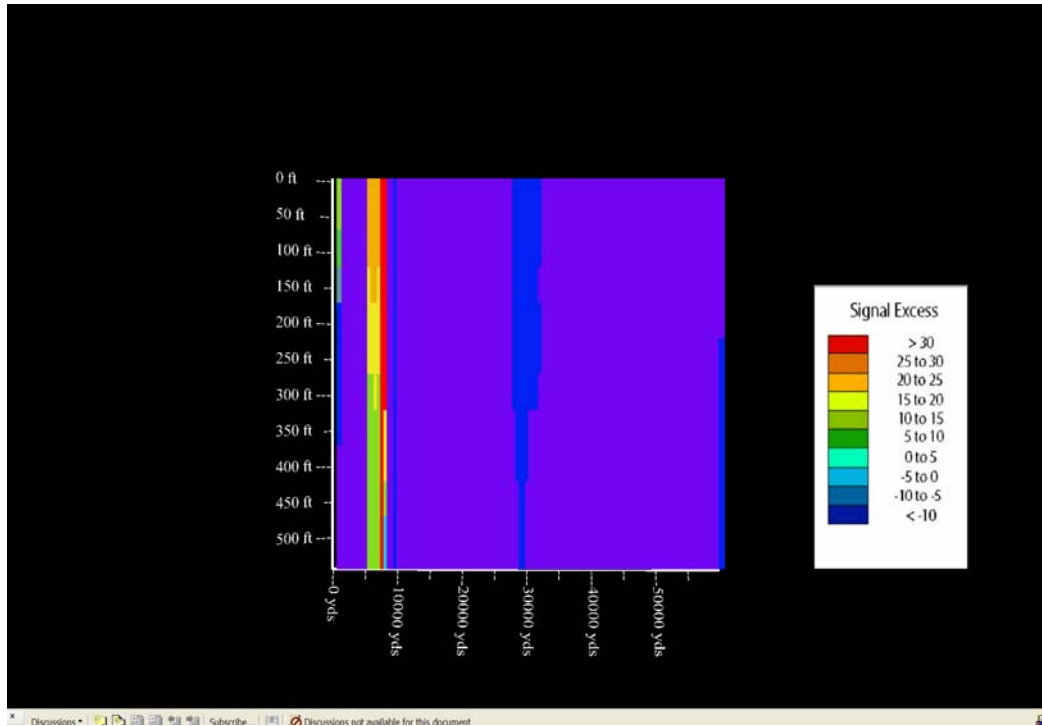
Figure 44.   The Full Field plot shows signal excess values as a function of depth and
        range.

The full field plot is generated using the LineOfBearingPlot class. The class
accepts a RadialBlade that corresponds to the closest radial to the requested bearing. The
RadialBlade objects possess signal excess values per depth and range for a specified
radial. The LineOfBearingPlot class then formulates a well-formed X3D scene. The plot
itself is created by creating an IndexedFaceSet for each signal excess point. In this
example, there are 10 depths and 121 ranges for a total of 1210 signal excess points. The
discrete ranges for each value point are specified in the CASS/GRAB model output file.
In this example, the greatest range is approximately 60,000 yards from the source. As the
12 IndexedFaceSets are created, each of the 121 data points (one for each range) of each
radial are tested in a switch statement to determine the color to be rendered.

E. SUMMARY

This chapter discusses some of the challenges associated with sonar visualization, and then describes some of the steps taken in this research aimed at overcoming these obstacles. The Sonar-Visualization API is then described to include its various 2D and 3D plots and scenes.

THIS PAGE INTENTIONALLY LEFT BLANK

X. CONCLUSIONS AND RECOMMENDATIONS

A. CONCLUSIONS

As the U.S. Navy steadily moves toward implementing a net-centric architecture, it would be foolish to forego this opportunity to inject modeling and simulation resources into the tactical picture. Military networks are now secure and reliable enough to pass tactically relevant data between systems and platforms. During Operation Enduring Freedom, the nature of naval warfare began to shift behind the scenes. Ships began to coordinate with other ships and shore-base assets using chat. Updated Air Tasking Orders (ATOs) were quickly disseminated via email or posted on a SIPRNET web page. Similar existing technologies such as XML Tactical Chat (XTC) can be leveraged to bring even more capabilities to the warfighter.

Modeling and simulation is recognized as an important tool for training and planning in many fields. In certain arenas, such as undersea warfare (USW), it is simply vital to success. The incredibly complex and variable conditions of the ocean make it nearly impossible to effectively deploy USW assets without any knowledge of predicted ranges and sonar performance. Unfortunately, in the past such information was often not available in a real-time environment. Some programs, such as PC-IMAT, are making strides to provide sonar prediction capability using the improved processing power of today's laptop or desktop. Yet, there are many other models out there that perform better in certain environments and many other shore-based computing assets that are underutilized in live battlespace. An ideal system ought to be able to choose the model best suited for a particular scenario, and then use the results to visualize sonar performance in a way that is uniform, sharable, and intuitive.

These goals can be accomplished through a net-centric sonar visualization architecture that is governed by a common XML standard. By developing a common ontology for all USW sensors, environmental data, prediction models, and visualization tools; all sonar modeling assets will be able to "plug-in" to each other and be available where they are needed, in the hands of the warfighter. This thesis demonstrates how such capabilities are possible. Specifically, A preliminary schema developed in this thesis is

107

designed to specify the capabilities of the RRA and CASS-GRAB sonar prediction models (the two available to NPS at the time) and FnMOC Metcast environmental data. In no way is this schema expected to be a universal ontology, but merely an exemplar that might show how a schema can link diverse stages of the sonar modeling process together. In addition, the Metcast Web Service and the Sonar Visualization API were created to demonstrate how the schema can be used implemented to exercise this architecture.

Two exemplars were created in conjunction with this thesis. The first was the AUV Workbench's Sonar Visualization Workbench Panel (SVWP). The SVWP's graphical user interface allows a user to manually work through the sonar modeling process. A user enters a few specific parameters and then manipulates buttons to access environmental data via the MWS, choose and run a sonar model either locally or remotely, then visualize results in an embedded window using the Sonar Visualization API. The second exemplar is the WEAVER site. The WEAVER site was developed in partnership with Sonalysts Inc. under NAVAIR funding. It is a web page located on the SIPRNET that allows a user to remotely access high-resolution sonar prediction resources via a simple Web browser. After entering a small set of tactical parameters into a web form, the user clicks a button on the Web page and the Web server orchestrates the sonar modeling process by calling MWS, running a sonar prediction model and then displaying multiple results in the native browser as various 2D plots or an X3D scene.

The Rim of the Pacific Exercise (RIMPAC) 2004 was held in July and August of 2004. Both the WEAVER site and the AUV Workbench were demonstrated offline at most participating facilities in Hawaii. While the two exemplars obviously didn't receive high marks as a finished product, most of those interviewed instantly recognized the potential that future applications based on such an architecture would have. The WEAVER project and the AUV Workbench will both continue development and the WEAVER project is expected to be operational and on display during the Undersea Dominance 2004 exercise. It is clear that Web-based technologies can play an integral role in bringing modeling and simulation resources to the warfighter.

B.   RECOMMENDATIONS FOR FUTURE WORK

1.   Sonar Modeling Schema

The Sonar Modeling Schema as is stands currently is only designed to accommodate the USW resources that were available to NPS inspection. This included the Metcast server at FnMOC, the RRA code developed at NPS by Dr. Ray Ziomek, and the CASS-GRAB model. Many other environmental tools, sonar models, and other resources need to be included in the schema before is becomes an ontology that is representative of all sonar modeling. Once this is accomplished, it can then be included in the schema currently being developed by the USW-XML Working Group and be proposed for implementation. The schemas developed in this thesis are necessarily narrow and only serve to demonstrate the capabilities inherent in such an architecture.

2.   RRA Sonar Server

As part of this thesis, the RRA Java code base developed by Timothy Holiday in 1998 was modified to ingest "live" environmental analysis and forecast data retrieved from MWS. This includes actual sound speed profiles and bathymetry data. Unfortunately, it was not within the scope of this thesis to fully integrate RRA into the net-centric architecture. The RRA code needs to also incorporate significant wave height and wind characteristics to take into account surface interaction. Also, it needs to be modified to use picking algorithms to conduct collision detection to more accurately determine and model bottom and surface interaction. This collision detect can also be used to determine target detection and vulnerability. In addition, due to RRA's ray tracing nature, it does not directly provide signal excess values as a function of predetermined ranges and locations as the architecture calls for (and as is returned by most prediction models such as CASS-GRAB). The picking algorithm can be used to detect collision detection at these predetermined points and record signal excess values. Once implemented, the RRA code will be fully compliant with the Sonar Modeling Schema and interchangeable with other sonar modeling tools.

3. AUV Workbench Sonar Visualization Panel

The Sonar Visualization Workbench Panel (SVWP) in the AUV Workbench can greatly benefit from continued work. Currently, the panel is embedded in the AUV Workbench as a related but independent tool. It provides value by imparting tactical information to the user about the potential operating environment. One of the next steps in AUV Workbench development is to integrate the sonar prediction data into the AUV physical model and incorporate predicted sonar detection into the AUV scene. Operators will be able to visualize coverage areas, detection zones, and possible target detection as the AUV progresses through its mission. In addition, as more sonar models and prediction tools become available, the panel will include them as options. Comparisons between model outputs will yield further visualizations and insight that were not previously possible.

4. CASS-GRAB Output/Improved 3D Visualizations

The fields specified in the XML output returned from the CASS-GRAB sonar model mirrors the output returned from the actual model. Unfortunately, the native nomenclature used by the CASS-GRAB model does not translate well to a universal and intuitive description language. For example, the CASS-GRAB field "TIME_RATIO" corresponds to signal processing ratios needed to calculate signal excess values. The CASS-GRAB supporting XML utilities need to convert the CASS-GRAB output files to a more intuitive and universal namespace design. Ultimately, this design should match the Sonar Modeling Schema.

In addition, these XML output files are currently being converted into X3D visualizations by the Sonar Visualization API. These visualizations include bullseye, full field, and stack scene plots. This suite of visualizations needs to be expanded to include many more valuable plots. Three-dimensional isosurface plots depicting signal excess, transmission loss, and sound speed are a few potential examples.

5.   CASS/GRAB Exposed as Web service

Unlike RRA, NPS does not possess CASS-GRAB code and thus has not implemented it as a Web service. As part of the WEAVER project, Sonalysts Inc. has exposed CASS-GRAB to the WEAVER Web server via an XML wrapper. The next step towards making CASS-GRAB freely available to Sonar Modeling Schema compliant users/clients/assets is exposing it as a Web service. Once this is done, any client implementation will be able to access CASS-GRAB for its sonar prediction model needs.

6.   Scaling Up and Out

A great deal of important future work is now possible. The capabilities demonstrated in this thesis can serve as the foundation for further visualization techniques and adding/extending computational sonar models. Most importantly, at-sea testing with fleet operators needs to continue in order to maximize tactical capabilities.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX A. METCASTBEAN.JAVA

```
/* _____

   Title:          MetcastBean.java

   Description:     Metcast Web Service (MWS) class that serves as WS interface.

   Created:         March 31, 2004, 10:54 AM

   Revised:         31 March 2004

   Project:         FnMOC Metcast Web Service

                    Naval Postgraduate School, Monterey, CA

   Compiler:        JDK 1.4.2

   Author:          Scott Rosetti, LT USN

   Version:         1.0

   _____

*/


package nps.navy.mil.xmsf.MWS.MetcastTools;

import java.io.*;

import java.util.*;

/**

 *  This bean acts as web service interface for the Metcast Web Service (MWS). This class accepts

 *  SOAP-XML request from a client with latitude. longitude and size of area. It will then call appropriate methods to

 *  form Metcast Brokering Language request to Metcast Server and transform response into XML format.

 *

 */

public class MetcastBean {


  StringBuffer results = new StringBuffer();

  private static final double NM_TO_DEGREES = (1/60);


  /** Creates a new instance of MetcastBean */

  public MetcastBean() {

  }


  /** The MetcastBean class accepts latitude (double), longitude (double) and size of area (one side - double) and

   *  instantiates and calls helper classes to generate XML response to query (Includes ssp, sea temp,

   *  bathymetry, wind speed components and significant wave height.

   */

  public String makeRequest(double latitude, double longitude, double size) throws Exception {


    // check to ensure for valid latitude and longitude
```

```java
if ((latitude > 90) || (latitude < -90) || (longitude > 180) || (longitude < -180)) {

    results.append("<MetcastResponse>\n\t<ERROR msg=\"ERROR: Invalid Lat/Long entered.\"/>\n</MetcastResponse>\n");

}
// check to ensure valid size
else if (size <= 0) {

    results.append("<MetcastResponse>\n\t<ERROR msg=\"ERROR: Invalid size entered.\"/>\n</MetcastResponse>\n");

}
else {

    FetchGridValuesArea area = new FetchGridValuesArea();
    double deg = size*NM_TO_DEGREES;


    results.append(area.makeRequest(latitude, longitude, deg));

}
return results.toString();
}//end makeRequest
}//end MetcastBean
```

APPENDIX B. FETCHGRIDLISTAREA.JAVA

```
/* _____

   Title:        FetchGridListArea.java

   Description:  Access Metcast data via web service and convert to XML

   Created:      March 31, 2004, 10:54 AM

   Revised:      31 March 2004

   Project:      FnMOC Metcast Web Service

                 Naval Postgraduate School, Monterey, CA

   Compiler:     JDK 1.4.2

   Author:       Created by Ryan Hofschneider (FnMOC) modified by Scott Rosetti, LT USN (NPS)

   Version:      1.0

   _____

*/


package nps.navy.mil.xmsf.MWS.MetcastTools;

import java.util.Iterator;

import java.util.ArrayList;

import java.util.Collection;

import java.util.Calendar;

import java.util.Enumeration;

import java.net.URL;

import java.net.MalformedURLException;

import HTTPClient.ModuleException;

import HTTPClient.ParseException;

import HTTPClient.ProtocolNotSuppException;

import java.io.IOException;

import java.util.logging.Logger;

import java.util.logging.Level;

import java.util.logging.SimpleFormatter;

import java.util.logging.FileHandler;

import java.util.logging.Handler;

import HTTPClient.HTTPConnection;

import HTTPClient.HTTPResponse;

import HTTPClient.NVPair;

import java.io.BufferedInputStream;

import java.util.Map;

import java.util.HashMap;

import mil.navy.fnmoc.commons.mimeparser.MultipartReader;

import mil.navy.fnmoc.commons.mimeparser.MimeItemException;
```

```java
/**
 *  This class makes a properly formatted request to a specified Metcast Channels
 *  server using the Metcast Brokering Lanuage (MBL) based on inputs provided
 *  via the makeRequest method.
 */
public class FetchGridValuesArea
{
    private static final String MBL_AREA_ID = "NPS-Ryan";
    private static final String METCAST_SERVER_URL_SPEC =
        "http://metcast.metnet.navy.mil/cgi-bin/mcsrvr/server";


    private static final Logger logger =
        Logger.getLogger(FetchGridValuesArea.class.getName());
    private static final String LOGFILE_PATH = "%t/FetchGridValues.log";


    /** This method defined the request parameters including the models to be
     *  queried, what products are to be returned, and what depths (levels) to
     *  be returned. These product descriptions are currently "hardcoded" but
     *  should eventually be determined dynamically.
     */
    private static ArrayList createProductList()
    {
        /* Permitted range of values for query determined from
         * potential table of contents. For example purposes I've
         * hard-coded them. You could read them from a properties file
         * Or fetch the potential table of contents from the Metcast
         * server, and dynamically create your product list.
         */

        double[] otisDepths = new double[] {
            0f, 2.5f, 7.5f, 12.5f, 17.5f, 25f, 32.5f, 40f, 50f, 62.5f, 75f,
            100f, 125f, 150f, 200f, 300f, 400f, 500f, 600f, 700f, 800f, 900f,
            1000f, 1100f, 1200f, 1300f, 1400f, 1500f, 1750f, 2000f, 2500f,
            3000f, 4000f, 5000f
        };

        /* OTIS model generates values for a "snapshot" in time; there are
         * no taus. Use the AnalysisProduct class instead of the ForecastProduct
         * class to generate MBL request that won't specify a specifc valid time.
         */
        Product soundSpeed  = new AnalysisProduct("OTIS_GLOBAL", "",
                                "Sound Speed",
                                "dpth_sfc", otisDepths);
```

116

```java
        Product seaTemperature = new AnalysisProduct("OTIS_GLOBAL", "",
                                "Sea-water Temperature",
                                "dpth_sfc", otisDepths);


        /* The NOGAPS model has the concept of different runtimes, if you don't
         * specify "real" as the runtime, the query may also return results
         * from a "prelim" or "post"  model run which doesn't have as many taus.
         */
        Product windVelocityU = new ForecastProduct("NOGAPS", "real",
                                "Wind Velocity U-Component",
                                "ht_sfc", new double[] { 10 });


        Product windVelocityV = new ForecastProduct("NOGAPS", "real",
                                "Wind Velocity V-Component",
                                "ht_sfc", new double[] { 10 });


        /* Significant wave height is at the surface of the ocean, no level values
         * are necessary, hence the empty level array.
         */
        Product sigWaveHeight = new ForecastProduct("WW3_GLOBAL", "",
                                "Significant Wave Height",
                                "surface", new double[] {});


        Product bathy = new AnalysisProduct("ETOPO2", "",
                          "Seafloor and Land Elevation",
                          "surface", new double[] {});


        ArrayList products = new ArrayList();
        products.add(soundSpeed);
        products.add(seaTemperature);
        products.add(windVelocityU);
        products.add(windVelocityV);
        products.add(sigWaveHeight);
        products.add(bathy);


        return products;
    }// end createProductList


    /** This method creaes the MBL string based of input paramters that include
     *  geographic box points, time, and the desired products)
     */
    private static String createMBL(double n, double w, double s, double e,
```

```java
                    Calendar validTime, Collection products)
{
    Iterator i = products.iterator();
    StringBuffer mbl = new StringBuffer("(" + MBL_AREA_ID + " (products \n");
    while (i.hasNext()) {
        Product p = (Product)i.next();
        /* Valid time ignored for AnalysisProduct when constructing MBL */
        mbl.append(p.toMBL(n, w, s, e, validTime));
        if (i.hasNext()) {
            mbl.append("\n");
        }
    }
    mbl.append("))");

    return mbl.toString();
} // end createMBL


/** This method posts the message via HTTP and retrieves the response from
 *  the Metcast server.
 */
private static GridResultListArea retrieveGrids(String mbl, URL serverURL)
{
    GridResultListArea results = new GridResultListArea();

    try {
        /* http://www.innovation.ch/java/HTTPClient/ */
        HTTPConnection connection = new HTTPConnection(serverURL);
        NVPair headers[] = { new NVPair("Content-type", "text/x-mbl") };

        HTTPResponse response =
            connection.Post(serverURL.getFile(), mbl, headers);

        if (response.getStatusCode() >= 400) {
            logger.severe("Error connecting to server: " +
                response.getReasonLine());
            logger.fine("Error text: " + response.getText());
        } else if (response.getStatusCode() >= 300) {
            logger.warning("Server does not have data matching request");
        }
        else {
            /* Unpack multi-part mime response */
            Map responseHeaders = new HashMap();
            Enumeration e = response.listHeaders();
```

```java
        while (e.hasMoreElements()) {
            String item = (String) e.nextElement();
            logger.finest("Header: " + item + ":" +
                response.getHeader(item));
            responseHeaders.put(item.toLowerCase(),
                response.getHeader(item));
        }

        BufferedInputStream instream =
            new BufferedInputStream(response.getInputStream());
        MultipartReader reader =
            new MultipartReader(responseHeaders, instream, results);

        reader.readMimeEntry();
    }
} catch (ModuleException e) {
    logger.log(Level.SEVERE, "", e);
} catch (ParseException e) {
    logger.log(Level.SEVERE, "", e);
} catch (MimeItemException e) {
    logger.log(Level.SEVERE, "", e);
} catch (ProtocolNotSuppException e) {
    logger.log(Level.SEVERE, "", e);
} catch (IOException e) {
    logger.log(Level.SEVERE, "", e);
}

return results;
} // end retreiveGrids

/** log used for troubleshooting and reference */
private static void createLog() throws IOException
{
    logger.setUseParentHandlers(false);
    Handler h = new FileHandler(LOGFILE_PATH);
    h.setFormatter(new SimpleFormatter());
    logger.addHandler(h);
    logger.setLevel(Level.FINEST);

    Logger mrl = Logger.getLogger("mil.navy.fnmoc.commons.MimeReader");
    mrl.addHandler(h);
    mrl.setLevel(Level.FINEST);
    mrl = Logger.getLogger(GridResultListArea.class.getName());
```

119

```java
        mrl.addHandler(h);
        mrl.setLevel(Level.FINEST);
    } // end createLog


    /** NPS method that orchestrates the Metcast request-response sequence */
    public String makeRequest(double lat, double lon, double size) {

        double n = lat;
        double w = lon;
        double s = lat-size;
        double e = lon+size;

        // ***check with FnMOC for compatibility*** if the eastern edge is
        // greater than 180, then set eastern edge to be equal to -180 plus the
        // remaining difference in size
        if (lon > (180-size)) { e = -180 + (size -(180 - lon)); }

        Calendar validTime = Calendar.getInstance();
        StringBuffer xmlRep = new StringBuffer("<?xml version='1.0' encoding='UTF-8'?>\n<MetcastResponse>\n");

        xmlRep.append("\t<Time>" + validTime.getTime() + "</Time>\n");
        xmlRep.append("\t<Location>\n\t\t<Latitude>" + lat + "</Latitude>\n\t\t<Longitude>" + lon +
                                                        "</Longitude>\n\t</Location>\n");

        try {
            createLog();
            ArrayList products = createProductList();
            String mbl = createMBL(n, w, s, e, validTime, products);

            logger.info("\n" + mbl);

            GridResultListArea results = retrieveGrids(mbl, new URL(METCAST_SERVER_URL_SPEC));

            Iterator i = results.iterator();
            while (i.hasNext()) {
                xmlRep.append(((GridResultArea)i.next()).dump());
            }

            if (products.size() > results.size()) {
                String error =
                    "Fetched " + results.size() +
                    " of " + products.size() + " requested products. " +
                    "See log file in /tmp.";
```

120

```
        logger.severe(error);

        System.err.println(error);

      }


    } catch (Exception err) {

      System.out.println("ERROR: " + err);

    }

    xmlRep.append("</MetcastResponse>");


    return xmlRep.toString();

  } // end makeRequest

} // end FetchGridValues}
```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX C. METCASTCLIENTMAIN.JAVA

```
/* _____

   Title:         MetcastClientMain.java

   Description:   Utility class that parses an XML document returned from Metcast Web   Service.

   Created:       April 19, 2004, 10:54 AM

   Revised:       27 April 2004

   Project:       Tactical Web Services - Sonar Visualization

                  Naval Postgraduate School, Monterey, CA

   Compiler:      JDK 1.4.2

   Author:        Scott Rosetti, LT USN

   Version:       1.0

   _____

*/

package nps.navy.mil.xmsf.SonarVis.Metcast.MetcastTools;


//standard packages

import java.io.*;

import java.net.*;

import java.util.*;

import java.awt.*;

import javax.swing.*;


// JAXP parsers

import org.xml.sax.*;

import org.xml.sax.helpers.DefaultHandler;

import javax.xml.parsers.SAXParserFactory;

import javax.xml.parsers.ParserConfigurationException;

import javax.xml.parsers.SAXParser;


/**

 * This class orchestrates the Web service call and parses the XML

 * response and instatiates an EnvironmentalData ojbject to contain it.

 */

public class MetcastClientMain extends DefaultHandler {

  /** directory location of native XML files */

  private static final String XML_HOME = "build/MetcastFiles/XML/";


  /** url location of Metcast Web Service */

  private static final String METCAST_ADDRESS = "http://xchat.movesinstitute.org:8080/soap/servlet/rpcrouter";
```

```java
/** container to hold returned environmental data */
private static EnvironmentalData data;

/** time of request */
private String time;

/** latitude of metcast request */
private double latitude;

/** longitude of metcast request */
private double longitude;

/** size of area requested */
private double boxSize;

/** wind speed u-component (north => south) */
private double uComponent;

/** wind speed v-component (west => east) */
private double vComponent;

/** significant wave height */
private double waveHeight;

/** depths of sound speed data points */
private String soundLevels;

/** sound speed values */
private String soundValues;

/** depths of sea temperature values */
private String seaLevels;

/** sea temperature values */
private String seaValues;

/** bathymetry values */
private String bathValues;

/** resolution of bathymetry data (in degrees) */
private double resolution;
```

```java
    /** sound speed profile object */
    private SoundSpeed ssp;

    // parsing variables */
    private StringBuffer textBuffer;
    private String element;
    static private Writer  out;
    private boolean startFlag;
    private boolean isDoneParsing;
    private boolean isValid = true;

    /** Creates a new instance of SAXParser */
    public MetcastClientMain() {
        isDoneParsing = false;
        time = "failed response";
        latitude = 0.0;
        longitude = 0.0;
        uComponent = 0;
        vComponent = 0;
        waveHeight = 0;
        soundLevels = "0";
        soundValues = "0";
        seaLevels = "0";
        seaValues = "0";
        resolution = 0.033333333;
    }



// output header
    public void startDocument() throws SAXException
    {
        echo("<?xml version='1.0' encoding='UTF-8'?>");
        endLine();
    }

    // flush debugging output
    public void endDocument()throws SAXException {
        try {
            endLine();
            out.flush();

            processData();
```

125

```java
    } catch (IOException e) {
        throw new SAXException("I/O error", e);
    }
}


public void startElement(String namespaceURI, String sName, String qName,
                Attributes attrs)throws SAXException {

    echoText();
    String eName = sName;

    if ("".equals(eName)){
        eName = qName;
    }

    // print tag
    echo("<"+eName);

    element = eName;
    startFlag = true;

    // print attributes if any
    if (attrs != null) {
        for (int i = 0; i < attrs.getLength(); i++) {
            String aName = attrs.getLocalName(i); // Attr name
            if ("".equals(aName)) aName = attrs.getQName(i);
            echo(" ");
            echo(aName+"=\""+attrs.getValue(i)+"\"");
            if (aName.equals("Resolution")) {
                resolution = new Double(attrs.getValue(i)).doubleValue();
        }
            if (aName.equals("msg")) {
                isValid = false;
                System.out.println(attrs.getValue(i));
        }
        }
    }
    echo(">");
}

public void endElement(String namespaceURI, String sName, String qName) throws SAXException {

    // prints element value if any
```
126

```java
        echoText();

        String eName = sName;
        if ("".equals(eName)) {
            eName = qName;
        }

        // print closing tag
        echo("</"+eName+">");

        startFlag = false;
    }

    public void characters(char buf[], int offset, int len)
    throws SAXException
    {
        String s = new String(buf, offset, len);
        if (textBuffer == null) {
            textBuffer = new StringBuffer(s);
        } else {
            textBuffer.append(s);
        }
    }


    private void echoText() throws SAXException
    {
        if (textBuffer == null) return;
            String s = ""+textBuffer;

        //*******************************************
        // Set Properties
        //*******************************************

        if (element.equals("Time") && startFlag == true) {
                time = s.toString();
        }
        if (element.equals("Latitude") && startFlag == true) {


                latitude = new Double(s).doubleValue();
        }
        if (element.equals("Longitude") && startFlag == true) {
```

127

```java
        longitude = new Double(s).doubleValue();

  }
  if (element.equals("SoundSpeedLevels") && startFlag == true) {


        soundLevels = s.toString();
  }
  if (element.equals("SoundSpeedValues") && startFlag == true) {


        soundValues = s.toString();
  }
  if (element.equals("SeawaterTemperatureLevels") && startFlag == true) {


        seaLevels = s.toString();
  }
  if (element.equals("SeawaterTemperatureValues") && startFlag == true) {


        seaValues = s.toString();
  }
  if (element.equals("WindComponent_UValues") && startFlag == true) {


        uComponent = (new Double(s)).doubleValue();
  }
  if (element.equals("WindComponent_VValues") && startFlag == true) {


        vComponent = (new Double(s)).doubleValue();
  }
  if (element.equals("SignificantWaveHeightValues") && startFlag == true) {


        waveHeight = (new Double(s)).doubleValue();
  }
  if (element.equals("BathymetryValues") && startFlag == true) {


        bathValues = s.toString();
  }
     echo(s);
     textBuffer = null;
}


private void echo(String s)
throws SAXException
{
  try {
```

```java
      out.write(s);

      out.flush();

    } catch (IOException e) {

      throw new SAXException("I/O error", e);

    }

}


// Start a new line

private void endLine()

throws SAXException

{

   String lineEnd =  System.getProperty("line.separator");

   try {

      out.write(lineEnd);

   } catch (IOException e) {

      throw new SAXException("I/O error", e);

   }

}


public void processData() {


   if(isValid) {


      data = new EnvironmentalData(time, latitude, longitude, uComponent, vComponent, waveHeight, soundLevels, soundValues,
                  seaLevels, seaValues, bathValues, resolution);


      isDoneParsing = true;


      System.out.println("Ready to Display Data......");

   }


}// end processData

public EnvironmentalData parseDocument(double lat, double lon, double size) {


   boxSize = size;

   latitude = lat;

   longitude = lon;


   try {

      URL server = new URL(METCAST_ADDRESS);

      MetcastTestClient client = new MetcastTestClient(lat, lon, size);

      String results = client.getMetcastData(server);
```

129

```java
        File file = new File(XML_HOME + "MetcastTest.xml");

        BufferedWriter writer = new BufferedWriter(new FileWriter(file));

        writer.write(results);

        writer.close();

      }catch(Exception e){

        System.out.println("ERROR: " + e);

      }


      // Use an instance of ourselves as the SAX event handler

      DefaultHandler handler = new MetcastClientMain();


      // Use the default (non-validating) parser

      SAXParserFactory factory = SAXParserFactory.newInstance();


      if (isValid) {


        try {

          // Set up output stream

          out = new OutputStreamWriter(System.out, "UTF8");


          // Parse the input

          SAXParser saxParser = factory.newSAXParser();

          saxParser.parse( new File(XML_HOME + "MetcastTest.xml"), handler);


        } catch (Throwable t) {

          t.printStackTrace();

        }


      } else {

        data = null;

      }

      return data;

  }// end parseDocument

}// end MetcastClientMain
```

130

APPENDIX D. METCASTTESTCLIENT.JAVA

```
/* _____

   Title:        MetcastTestClient.java

   Description:  Utility class that acts as the SOAP interface for the Metcast Web Service.

   Created:      April 19, 2004, 10:54 AM

   Revised:      27 April 2004

   Project:      Tactical Web Services - Sonar Visualization

                 Naval Postgraduate School, Monterey, CA

   Compiler:     JDK 1.4.2

   Author:       Scott Rosetti, LT USN

   Version:      1.0

   _____

*/

package nps.navy.mil.xmsf.SonarVis.Metcast.MetcastTools;

import java.io.*;

import java.net.*;

import java.util.*;

import java.awt.*;

import javax.swing.*;

import org.apache.soap.*;

import org.apache.soap.rpc.*;

import org.apache.soap.rpc.Call;

/**

 * This class interfaces with Web service via SOAP protocol

 */

public class MetcastTestClient {

  /** directory location of native XML files */
  private static final String XML_HOME = "build/MetcastFiles/XML/";

  /** url location of Metcast Web Service */
  private static final String METCAST_ADDRESS = "http://xchat.movesinstitute.org:8080/soap/servlet/rpcrouter";

  /** uniform resource name of Metcast Web Service */
  private static final String TARGET = "urn:FnMOC:MetcastServer";

  /** latitude of NW corner of box */
  private double latitude = 0.0;

  /** longitude of NW corner of box */
```

```java
private double longitude = 0.0;

/** size (in nm) of side of box */
private double s = 60.0;

/** Creates a new instance of MetcastTestClient */
public MetcastTestClient(double lat, double lon, double size) {
    latitude = lat;
    longitude = lon;
    s = size;
}// end contructor

//Makes SOAP call to Metcast WEb Service */
public String getMetcastData(URL url) throws Exception {

    String value = null;

    Double lat = new Double(latitude);
    Double lon = new Double(longitude);
    Double size = new Double(s);
    Calendar validTime = Calendar.getInstance();

    try {
        Call call = new Call();
        call.setTargetObjectURI(TARGET);
        call.setEncodingStyleURI(Constants.NS_URI_SOAP_ENC);

        call.setMethodName("makeRequest");
        Vector params = new Vector();
        params.addElement(new Parameter("latitude", Double.class, lat, null));
        params.addElement(new Parameter("longitude", Double.class, lon, null));
        params.addElement(new Parameter("size", Double.class, size, null));
        //params.addElement(new Parameter("time", Calendar.class, validTime, null));
        call.setParams(params);
        org.apache.soap.rpc.Response resp = call.invoke(url, "" );

        // Check the response.
        if ( resp.generatedFault() ) {
            Fault fault = resp.getFault ();
            System.out.println("The call failed: ");
            System.out.println("Fault Code   = " + fault.getFaultCode());
            System.out.println("Fault String = " + fault.getFaultString());
        }
```

132

```java
            Parameter result = resp.getReturnValue();
            //System.out.println(result);

            value = (String)result.getValue();

        } catch (Exception e) {

            System.out.println("ERROR: " + e);
        }
        return value;
    }//end getMetcastData
}// end MetcastTestClient
```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX E. ENVIRONMENTALDATA.JAVA

```
/* _____

   Title:        EnvironmentalData.java

   Description:  This class is a grouping class that holds specfic environmental variables

                 useful for sonar modeling (sound speed, sea temoerature, wind speed,

                 bathymetry, and significant wave height).

   Created:      April 19, 2004, 10:54 AM

   Revised:      27 April 2004

   Project:      Tactical Web Services - Sonar Visualization

                 Naval Postgraduate School, Monterey, CA

   Compiler:     JDK 1.4.2

   Author:       Scott Rosetti, LT USN

   Version:      1.0

   _____

*/

package nps.navy.mil.xmsf.SonarVis.Metcast.MetcastTools;

import nps.navy.mil.xmsf.SonarVis.Metcast.MetcastTools.Bathymetry;


/**

 * This class is a grouping class that holds specfic environmental variables useful for

 * sonar modeling (sound speed, sea temoerature, wind speed, bathymetry, and significant

 * wave height).

 */

public class EnvironmentalData {


  /** time of request */

  private String sampleTime = "none";


  /** latitude of NW corner of requested area*/

  private double latitude = 0.0;


  /** latitude of NW corner of requested area*/

  private double longitude = 0.0;


  /** wind speed u-component (north => south) */

  private double uComponent = 0;


  /** wind speed v-component (west => east) */

  private double vComponent = 0;
```

```java
/** significant wave height*/
private double waveHeight = 0;

/** depths of sound speed data points */
private String soundLevels = "none";

/** sound speed values */
private String soundValues = "none";

/** depths of sea temperature values */
private String seaLevels = "none";

/** sea temperature values */
private String seaValues = "none";

/** bathymetry values */
private String bathValues = "none";

/** resolution of bathymetry data (in degrees) */
private double resolution = 0.0333333;

/** wind speed object */
private Windspeed wind;

/** sound speed profile object */
private SoundSpeed ssp;

/** significant wave hieght object */
private SignificantWaveHeight waves;

/** sea temperature object */
private SeawaterTemp seaTemp;

/** bathymetry object */
private Bathymetry bathymetry;

/** Creates a new instance of EnvironmentalData */
public EnvironmentalData(String time, double lat, double lon, double u, double v, double wave, String soundLev, String soundVal,
String seaLev, String seaVal, String bathVal, double res) {

    sampleTime = time;
    latitude = lat;
    longitude = lon;
```

```java
        uComponent = u;
        vComponent = v;
        waveHeight = wave;
        soundLevels = soundLev;
        soundValues = soundVal;
        seaLevels = seaLev;
        seaValues = seaVal;
        bathValues = bathVal;
        resolution = res;
        System.out.println("Creating Parameter Objects ***************************");
        seaTemp = new SeawaterTemp(seaLevels, seaValues);
        waves = new SignificantWaveHeight(wave);
        bathymetry = new Bathymetry(bathVal);
        wind = new Windspeed(uComponent, vComponent);
        ssp = new SoundSpeed(soundLevels, soundValues);

    }// end constructor

    /** returns windspeed (u and v components)*/
    public double[] getWindSpeed() {

        double[] obj = new double[2];

        obj[0] = (double)wind.getDirectionDegrees();
        obj[1] = wind.getVelocity();

        return obj;
    }// end getWindSpeed

    /** calculates and returns vague wind direction */
    public String getVagueWindDirection() {

        String dir = new String();
        double speed = wind.getDirectionDegrees();

        if(speed >= 22.5 && speed < 67.5) {
            dir = "NE";
        }
        if(speed >= 67.5 && speed < 112.5) {
            dir = "East";
        }
        if(speed >= 112.5 && speed < 157.5) {
            dir = "SE";
```

137

```java
    }
    if(speed >= 157.5 && speed < 202.5) {
       dir = "South";
    }
    if(speed >= 202.5 && speed < 247.5) {
       dir = "SW";
    }
    if(speed >= 247.5 && speed < 292.5) {
       dir = "West";
    }
    if(speed >= 292.5 && speed < 337.5) {
       dir = "NW";
    }
    if(speed >= 337.5 || speed < 22.5) {
       dir = "North";
    }
    return dir;
}// end getVagueWindDirection

/** returns sound speed levels */
public double[] getSoundSpeedLevels() {

    return ssp.getLevelsInMeters();
}

/** returns sound speed values */
public double[] getSoundSpeedValues() {

    return ssp.getValuesInMetersPerSecond();
}

/** returns sea temp levels */
public double[] getSeaTempLevels() {

    return seaTemp.getTempLevels();
}

/** returns sea temp data values */
public double[] getSeaTempValues() {

    return seaTemp.getTempsInDegreesFarenheit();
}
```

```java
    /** returns bathymetry resolution */
    public double getResolution() {

        return resolution;

    }


    /** returns request time */
    public String getTime() { return sampleTime;}
    /** returns request latitude */
    public double getLatitude() { return latitude;}
    /** returns request longitude */
    public double getLongitude() { return longitude;}
    /** returns u component of wind speed */
    public double getUComponent() { return uComponent;}
    /** returns v component of wind speed */
    public double getVComponent() { return vComponent;}
    /** returns significant wave height */
    public double getWaveHeight() { return waveHeight;}
    /** returns sound speed depths */
    public String getSoundSpeedLevelString() { return soundLevels;}
    /** returns sound speed values */
    public String getSoundSpeedValueString() { return soundValues;}
    /** returns sea temp depths */
    public String getSeaTempLevelString() { return seaLevels;}
    /** returns sea temp values */
    public String getSeaTempValueString() { return seaValues;}
    /** returns bathymetry values */
    public String getBathymetryString() { return bathymetry.getBathymetry(); }
    /** returns sea state */
    public int getSeaState() { return waves.getSeaState(); }
    /** returns wind velocity */
    public double getWindVelocity() { return wind.getVelocity(); }
    /** return sound speed object */
    public SoundSpeed getSoundSpeed() { return ssp; }
    /** returns sea temp profile object */
    public SeawaterTemp getSeawaterTemp() { return seaTemp; }
    /** returns bathymetry object */
    public Bathymetry getBathymetry() { return bathymetry; }
}// end EnvironmentalData
```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX F. SONAROUTPUTMODELPARSER.JAVA

```
/* _____

   Title:         SonarModelOutputParser.java

   Description:   Parses output files generated by a sonar model engine into lists of

                  RadialBlades and DepthBlades and provides methods to generate specific 2D and 3D plots

                  (Currently supported visualizations are LineOfBearingPlot, BulleEyePlot, StackScene3D,

                  and Iso-Surface).

   Created:       May 20, 2004, 12:23 AM

   Revised:       14 June 2004

   Project:       Tactical Web Services - Sonar Visualization

                  Naval Postgraduate School, Monterey, CA

   Compiler:      JDK 1.4.2

   Author:        Scott Rosetti, LT USN

   Version:       1.0

   _____

*/

package nps.navy.mil.xmsf.SonarVis;


//standard packages

import java.io.*;

import java.net.URL;

import java.util.LinkedList;

import java.awt.*;

import javax.swing.*;


// JAXP parsers

import org.xml.sax.*;

import org.xml.sax.helpers.DefaultHandler;

import javax.xml.parsers.SAXParserFactory;

import javax.xml.parsers.ParserConfigurationException;

import javax.xml.parsers.SAXParser;


// local imports

import SonarVis.Metcast.MetcastTools.*;

import SonarVis.Metcast.MetcastUtilities.*;


/**

 * Parses output files generated by a sonar model engine into lists of

 * RadialBlades and DepthBlades.

 */
```

```java
public class SonarModelOutputParser extends DefaultHandler {

    private static final int DEPTH = 0;
    private static final int RADIAL = 1;
    private static final int MAX_ENTRY_SIZE = 10;


    /** holds model output data organized by depth */
    private DepthBlade blade;


    //private static SceneGenerator scene;
    /** object holds environmental data to include bathymetric data, ssp, and wind */
    private static EnvironmentalData data;


    /** array of lists
     *  [0] - RadialBlades
     *  [1] - DepthBlades
     */
    private static LinkedList [] blades;


    /** List of DepthBlades for use in creating visualizations */
    private static LinkedList depthBlades;


    /** utility list used in various methods */
    private static LinkedList temp;


    /** array of depths (in KM) */
    private static double [] depths;


    /** index to keep track of DepthBlades */
    private static int depthIndex;


    /** index to keep track of RadialBlades */
    private static int radialIndex;


    /** index to keep track of signal excess */
    private static int seIndex;


    /** array of filenames (sonar model output) to be parsed */
    private static String [] files;


    /** number of ranges per DepthBlade/RadialBlade */
    private static int numRanges;
```

142

```java
/** array of ranges (in KM) */
private static double [] ranges;

/** array of signal excess values */
private static double [][] excess;

/*8 index to keep track of ranges */
private static int rangeIndex;

/** number of radials per RadialBlade */
private static int numRadials;

/** bearing of closest radial blade */
private double bladeBearing;

/** boolean that keeps track of whether live Metcast data is used */
private boolean liveData;

/** closest data-laden bearing to requested bearing */
private double closestBearing;

// variables used for actual file parsing
private StringBuffer textBuffer;
private String element;
static private Writer  out;
private boolean startFlag;
private boolean isDoneParsing;

/** converts feet to kilometers */
private static final double FEET_TO_KM = .0003048;

/** converts kilometers to feet */
private static final double KM_TO_FEET = 3280.839895013123;

/** default bathymetry string (taken from Hawaiian OPAREA: 15, -145, 60 NM)*/
private String bath60 = "-5327.0 -5373.0 -5386.0 -5342.0 -5273.0 -5223.0 -5214.0 " +
            "-5214.0 -5219.0 -5262.0 -5311.0 -5318.0 -5254.0 -5135.0 " +
            "-5004.0 -4920.0 -4922.0 -4972.0 -5017.0 -5052.0 -5089.0 " +
            "-5114.0 -5109.0 -5089.0 -5078.0 -5073.0 -5057.0 -5048.0 " +
            "-5050.0 -5053.0 -5116.0 -5278.0 -5277.0 -5255.0 -5218.0 " +
            "-5189.0 -5186.0 -5206.0 -5216.0 -5238.0 -5284.0 -5325.0 " +
            "-5330.0 -5264.0 -5134.0 -4984.0 -4883.0 -4878.0 -4940.0 " +
            "-4996.0 -5026.0 -5081.0 -5128.0 -5124.0 -5093.0 -5082.0 " +
```

"-5083.0 -5064.0 -5083.0 -5074.0 -5043.0 -5132.0 -5236.0 " +
"-5216.0 -5188.0 -5157.0 -5156.0 -5190.0 -5218.0 -5234.0 " +
"-5249.0 -5274.0 -5309.0 -5318.0 -5260.0 -5138.0 -5003.0 " +
"-4916.0 -4907.0 -4957.0 -5003.0 -5040.0 -5111.0 -5160.0 " +
"-5164.0 -5104.0 -5081.0 -5111.0 -5091.0 -5124.0 -5068.0 " +
"-4991.0 -5066.0 -5201.0 -5177.0 -5167.0 -5161.0 -5172.0 " +
"-5193.0 -5211.0 -5213.0 -5200.0 -5210.0 -5248.0 -5270.0 " +
"-5232.0 -5136.0 -5047.0 -4998.0 -4984.0 -5005.0 -5024.0 " +
"-5043.0 -5130.0 -5182.0 -5167.0 -5088.0 -5044.0 -5096.0 " +
"-5115.0 -5106.0 -5043.0 -5002.0 -5038.0 -5219.0 -5208.0 " +
"-5209.0 -5203.0 -5187.0 -5168.0 -5170.0 -5151.0 -5110.0 " +
"-5111.0 -5148.0 -5177.0 -5160.0 -5113.0 -5084.0 -5078.0 " +
"-5065.0 -5054.0 -5059.0 -5062.0 -5133.0 -5169.0 -5123.0 " +
"-5066.0 -5005.0 -5030.0 -5071.0 -5054.0 -5000.0 -4977.0 " +
"-5015.0 -5295.0 -5283.0 -5273.0 -5243.0 -5183.0 -5094.0 " +
"-5053.0 -5035.0 -5005.0 -5020.0 -5062.0 -5086.0 -5059.0 " +
"-5051.0 -5087.0 -5127.0 -5120.0 -5074.0 -5067.0 -5071.0 " +
"-5112.0 -5149.0 -5147.0 -5130.0 -5083.0 -5075.0 -5074.0 " +
"-5041.0 -4990.0 -4966.0 -5001.0 -5402.0 -5377.0 -5348.0 " +
"-5297.0 -5216.0 -5094.0 -4994.0 -4954.0 -4951.0 -4970.0 " +
"-5037.0 -5077.0 -5027.0 -5024.0 -5081.0 -5129.0 -5111.0 " +
"-5042.0 -5009.0 -5033.0 -5119.0 -5189.0 -5203.0 -5192.0 " +
"-5155.0 -5130.0 -5099.0 -5058.0 -5018.0 -4997.0 -5013.0 " +
"-5457.0 -5430.0 -5392.0 -5358.0 -5315.0 -5252.0 -5197.0 " +
"-5102.0 -5023.0 -5007.0 -5049.0 -5092.0 -5059.0 -5048.0 " +
"-5079.0 -5099.0 -5063.0 -4985.0 -4932.0 -4968.0 -5085.0 " +
"-5182.0 -5195.0 -5163.0 -5126.0 -5099.0 -5079.0 -5060.0 " +
"-5044.0 -5040.0 -5044.0 -5369.0 -5373.0 -5362.0 -5361.0 " +
"-5324.0 -5315.0 -5350.0 -5271.0 -5163.0 -5109.0 -5086.0 " +
"-5086.0 -5072.0 -5072.0 -5098.0 -5107.0 -5060.0 -4965.0 " +
"-4894.0 -4901.0 -4972.0 -5049.0 -5075.0 -5054.0 -5021.0 " +
"-5003.0 -5009.0 -5036.0 -5057.0 -5077.0 -5088.0 -5181.0 " +
"-5233.0 -5221.0 -5228.0 -5193.0 -5262.0 -5378.0 -5342.0 " +
"-5266.0 -5191.0 -5136.0 -5108.0 -5092.0 -5102.0 -5159.0 " +
"-5193.0 -5141.0 -5045.0 -4960.0 -4907.0 -4917.0 -4968.0 " +
"-4988.0 -4969.0 -4933.0 -4914.0 -4934.0 -4995.0 -5054.0 " +
"-5100.0 -5123.0 -5020.0 -5050.0 -5034.0 -5053.0 -5074.0 " +
"-5194.0 -5324.0 -5305.0 -5240.0 -5175.0 -5138.0 -5130.0 " +
"-5106.0 -5119.0 -5251.0 -5340.0 -5297.0 -5225.0 -5151.0 " +
"-5081.0 -5047.0 -5040.0 -5024.0 -4987.0 -4935.0 -4901.0 " +
"-4909.0 -4967.0 -5039.0 -5090.0 -5114.0 -4934.0 -4922.0 " +
"-4917.0 -4942.0 -4988.0 -5070.0 -5152.0 -5161.0 -5126.0 " +
"-5099.0 -5086.0 -5116.0 -5080.0 -5104.0 -5286.0 -5385.0 " +

"-5388.0 -5360.0 -5318.0 -5274.0 -5213.0 -5150.0 -5116.0 " +
"-5083.0 -5028.0 -4967.0 -4945.0 -4972.0 -5026.0 -5065.0 " +
"-5078.0 -4893.0 -4866.0 -4857.0 -4872.0 -4904.0 -5007.0 " +
"-5080.0 -5041.0 -4992.0 -4987.0 -4952.0 -4961.0 -4993.0 " +
"-5061.0 -5208.0 -5299.0 -5354.0 -5364.0 -5325.0 -5259.0 " +
"-5204.0 -5187.0 -5191.0 -5177.0 -5127.0 -5071.0 -5037.0 " +
"-5031.0 -5047.0 -5057.0 -5058.0 -4853.0 -4819.0 -4805.0 " +
"-4817.0 -4826.0 -4954.0 -5047.0 -4967.0 -4883.0 -4920.0 " +
"-4885.0 -4808.0 -4874.0 -4970.0 -5097.0 -5192.0 -5250.0 " +
"-5263.0 -5225.0 -5153.0 -5123.0 -5156.0 -5197.0 -5218.0 " +
"-5194.0 -5148.0 -5116.0 -5101.0 -5091.0 -5076.0 -5057.0 " +
"-4800.0 -4759.0 -4744.0 -4762.0 -4796.0 -4906.0 -4961.0 " +
"-4910.0 -4804.0 -4783.0 -4788.0 -4741.0 -4794.0 -4887.0 " +
"-5003.0 -5105.0 -5168.0 -5179.0 -5140.0 -5088.0 -5067.0 " +
"-5093.0 -5152.0 -5189.0 -5189.0 -5179.0 -5175.0 -5177.0 " +
"-5155.0 -5110.0 -5061.0 -4772.0 -4719.0 -4703.0 -4727.0 " +
"-4778.0 -4833.0 -4814.0 -4779.0 -4731.0 -4699.0 -4707.0 " +
"-4708.0 -4760.0 -4849.0 -4950.0 -5034.0 -5080.0 -5083.0 " +
"-5061.0 -5054.0 -5036.0 -5028.0 -5075.0 -5128.0 -5154.0 " +
"-5159.0 -5168.0 -5186.0 -5171.0 -5122.0 -5056.0 -4831.0 " +
"-4746.0 -4724.0 -4753.0 -4824.0 -4924.0 -4897.0 -4801.0 " +
"-4742.0 -4711.0 -4705.0 -4717.0 -4768.0 -4847.0 -4927.0 " +
"-4981.0 -5002.0 -4998.0 -4988.0 -4995.0 -4989.0 -4982.0 " +
"-5013.0 -5052.0 -5077.0 -5096.0 -5123.0 -5144.0 -5139.0 " +
"-5097.0 -5025.0 -5031.0 -4858.0 -4810.0 -4813.0 -4885.0 " +
"-5015.0 -4989.0 -4908.0 -4851.0 -4819.0 -4801.0 -4800.0 " +
"-4825.0 -4873.0 -4921.0 -4946.0 -4952.0 -4953.0 -4957.0 " +
"-4959.0 -4960.0 -4969.0 -4983.0 -4996.0 -5003.0 -5016.0 " +
"-5040.0 -5067.0 -5074.0 -5044.0 -4985.0 -5370.0 -5023.0 " +
"-4896.0 -4869.0 -4843.0 -5016.0 -5114.0 -5103.0 -5073.0 " +
"-5044.0 -5009.0 -4969.0 -4943.0 -4941.0 -4946.0 -4945.0 " +
"-4950.0 -4971.0 -4996.0 -4988.0 -4983.0 -4991.0 -4981.0 " +
"-4973.0 -4970.0 -4976.0 -4993.0 -5013.0 -5022.0 -5006.0 " +
"-4962.0 -5894.0 -5627.0 -5596.0 -5586.0 -5581.0 -5615.0 " +
"-5503.0 -5423.0 -5375.0 -5336.0 -5292.0 -5218.0 -5131.0 " +
"-5074.0 -5041.0 -5021.0 -5024.0 -5065.0 -5104.0 -5101.0 " +
"-5079.0 -5032.0 -4985.0 -4974.0 -4979.0 -4989.0 -4999.0 " +
"-5009.0 -5014.0 -5000.0 -4960.0 -5987.0 -5908.0 -5872.0 " +
"-5874.0 -5946.0 -5980.0 -5882.0 -5785.0 -5714.0 -5656.0 " +
"-5575.0 -5456.0 -5352.0 -5293.0 -5248.0 -5209.0 -5194.0 " +
"-5187.0 -5184.0 -5177.0 -5111.0 -5020.0 -4983.0 -4999.0 " +
"-5028.0 -5045.0 -5050.0 -5049.0 -5046.0 -5028.0 -4983.0 " +
"-5988.0 -5998.0 -6038.0 -6028.0 -6061.0 -6128.0 -6081.0 " +

"-5995.0 -5919.0 -5857.0 -5783.0 -5688.0 -5603.0 -5548.0 " +
"-5507.0 -5459.0 -5406.0 -5348.0 -5215.0 -4893.0 -4791.0 " +
"-4951.0 -5017.0 -5074.0 -5124.0 -5148.0 -5146.0 -5138.0 " +
"-5127.0 -5101.0 -5048.0 -5909.0 -5945.0 -6049.0 -6198.0 " +
"-6231.0 -6164.0 -6108.0 -6043.0 -5982.0 -5929.0 -5874.0 " +
"-5819.0 -5787.0 -5784.0 -5783.0 -5753.0 -5703.0 -5656.0 " +
"-5628.0 -5637.0 -5523.0 -5319.0 -5240.0 -5257.0 -5291.0 " +
"-5299.0 -5290.0 -5278.0 -5265.0 -5241.0 -5183.0 -5742.0 " +
"-5744.0 -5731.0 -5699.0 -5773.0 -5904.0 -5961.0 -5969.0 " +
"-5949.0 -5913.0 -5871.0 -5848.0 -5866.0 -5917.0 -5964.0 " +
"-5972.0 -5946.0 -5913.0 -5900.0 -5900.0 -5781.0 -5577.0 " +
"-5462.0 -5448.0 -5483.0 -5512.0 -5513.0 -5497.0 -5473.0 " +
"-5434.0 -5365.0 -5570.0 -5596.0 -5665.0 -5659.0 -5644.0 " +
"-5720.0 -5802.0 -5858.0 -5873.0 -5844.0 -5801.0 -5794.0 " +
"-5846.0 -5937.0 -6021.0 -6063.0 -6065.0 -6060.0 -6061.0 " +
"-6022.0 -5935.0 -5815.0 -5700.0 -5662.0 -5686.0 -5720.0 " +
"-5733.0 -5722.0 -5697.0 -5660.0 -5611.0 -5479.0 -5487.0 " +
"-5473.0 -5456.0 -5496.0 -5574.0 -5664.0 -5740.0 -5770.0 " +
"-5745.0 -5701.0 -5698.0 -5763.0 -5869.0 -5964.0 -6016.0 " +
"-6033.0 -6060.0 -6137.0 -6271.0 -6221.0 -5984.0 -5821.0 " +
"-5761.0 -5788.0 -5845.0 -5880.0 -5877.0 -5844.0 -5803.0 " +
"-5768.0 -5444.0 -5437.0 -5422.0 -5399.0 -5418.0 -5463.0 " +
"-5537.0 -5614.0 -5650.0 -5629.0 -5586.0 -5584.0 -5650.0 " +
"-5748.0 -5831.0 -5870.0 -5874.0 -5878.0 -5896.0 -5883.0 " +
"-5871.0 -5841.0 -5762.0 -5730.0 -5772.0 -5849.0 -5904.0 " +
"-5905.0 -5861.0 -5808.0 -5783.0 -5427.0 -5378.0 -5372.0 " +
"-5390.0 -5404.0 -5417.0 -5443.0 -5490.0 -5525.0 -5515.0 " +
"-5488.0 -5487.0 -5533.0 -5611.0 -5680.0 -5711.0 -5702.0 " +
"-5679.0 -5673.0 -5680.0 -5689.0 -5671.0 -5620.0 -5605.0 " +
"-5655.0 -5740.0 -5803.0 -5803.0 -5749.0 -5687.0 -5666.0 " +
"-5429.0 -5419.0 -5416.0 -5406.0 -5393.0 -5384.0 -5404.0 " +
"-5436.0 -5455.0 -5460.0 -5452.0 -5449.0 -5459.0 -5493.0 " +
"-5548.0 -5589.0 -5582.0 -5539.0 -5503.0 -5496.0 -5509.0 " +
"-5500.0 -5464.0 -5447.0 -5486.0 -5560.0 -5619.0 -5628.0 " +
"-5579.0 -5520.0 -5503.0 -5403.0 -5433.0 -5446.0 -5426.0 " +
"-5396.0 -5372.0 -5381.0 -5414.0 -5438.0 -5436.0 -5416.0 " +
"-5400.0 -5407.0 -5436.0 -5475.0 -5516.0 -5524.0 -5492.0 " +
"-5457.0 -5441.0 -5442.0 -5427.0 -5386.0 -5350.0 -5355.0 " +
"-5388.0 -5435.0 -5464.0 -5440.0 -5398.0 -5382.0 -5367.0 " +
"-5393.0 -5411.0 -5414.0 -5396.0 -5384.0 -5397.0 -5421.0 " +
"-5429.0 -5414.0 -5382.0 -5358.0 -5358.0 -5389.0 -5438.0 " +
"-5480.0 -5490.0 -5466.0 -5432.0 -5431.0 -5434.0 -5398.0 " +
"-5340.0 -5283.0 -5262.0 -5293.0 -5355.0 -5396.0 -5397.0 " +

```
                    "-5380.0 -5363.0 ";

/** Creates a new instance of SonarModelOutputParser */
public SonarModelOutputParser(String[] files) {

    this.files = files;
    blades = new LinkedList[2];
    temp = new LinkedList();
    depthBlades = new LinkedList();
    depths = new double [files.length];
    depthIndex = 0;
    numRanges = 0;
    rangeIndex = 0;
    radialIndex = 0;
    seIndex = 0;
    liveData = false;
}


// output header
public void startDocument() throws SAXException
{
    echo("<?xml version='1.0' encoding='UTF-8'?>");
    endLine();
}


// flush debugging output
public void endDocument()throws SAXException {
    try {
        endLine();
        out.flush();

        processData();

    } catch (IOException e) {
        throw new SAXException("I/O error", e);
    }
}


public void startElement(String namespaceURI, String sName, String qName,
                Attributes attrs)throws SAXException {

    echoText();
    String eName = sName;
```

147

```java
        if ("".equals(eName)){
            eName = qName;
        }

        // print tag
        echo("<"+eName);

        element = eName;
        startFlag = true;

        // print attributes if any
        if (attrs != null) {
            for (int i = 0; i < attrs.getLength(); i++) {
                String aName = attrs.getLocalName(i); // Attr name
                if ("".equals(aName)) aName = attrs.getQName(i);
                echo(" ");
                echo(aName+"=\""+attrs.getValue(i)+"\"");

                if (aName.equals("time_idx") && startFlag == true) {
                    radialIndex = new Integer(attrs.getValue(i)).intValue();
                }

            }
        }
        echo(">");
    }

    public void endElement(String namespaceURI, String sName, String qName) throws SAXException {

        // prints element value if any
        echoText();

        String eName = sName;
        if ("".equals(eName)) {
            eName = qName;
        }
        // print closing tag
        echo("</"+eName+">");

        startFlag = false;
    }
```

148

```java
public void characters(char buf[], int offset, int len)
throws SAXException
{
  String s = new String(buf, offset, len);
  if (textBuffer == null) {
    textBuffer = new StringBuffer(s);
  } else {
    textBuffer.append(s);
  }
}

private void echoText() throws SAXException
{
  if (textBuffer == null) return;
    String s = ""+textBuffer;


  //*******************************************
  // Set Properties
  //*******************************************

  if (element.equals("RECEIVER_DEPTH") && startFlag == true) {
    depths[depthIndex] = new Double(s.toString()).doubleValue();
    depthIndex++;
  }
  if (element.equals("NUM_RANGES") && startFlag == true) {
    numRanges = new Integer(s.toString()).intValue();
    ranges = new double[numRanges];
  }
  if (element.equals("NUM_TIMES") && startFlag == true) {
    numRadials = new Integer(s.toString()).intValue();
    excess = new double[numRadials][numRanges];
  }
  if (element.equals("RANGE") && startFlag == true) {
    ranges[rangeIndex] = new Double(s.toString()).doubleValue();
    rangeIndex++;
  }
  if (element.equals("RATIO") && startFlag == true) {
    if (seIndex == numRanges) {
      seIndex = 0;
    }
    excess[radialIndex][seIndex] = convertToSignalExcess(new Double(s.toString()).doubleValue());
    seIndex++;
  }
```

149

```java
        echo(s);

        textBuffer = null;

    }


    private void echo(String s)
    throws SAXException
    {
        try {
            // out.write(s);
            out.flush();
        } catch (IOException e) {
            throw new SAXException("I/O error", e);
        }
    }


    // Start a new line
    private void endLine()
    throws SAXException
    {
        String lineEnd =  System.getProperty("line.separator");
        try {
            out.write(lineEnd);
        } catch (IOException e) {
            throw new SAXException("I/O error", e);
        }
    }


    /** method that will parse properly formatted files passed in and
     *  return array of Lists (RadialBlades and DepthBlades)
     */
    public LinkedList[] parseDocuments(String[] files) {

        DefaultHandler handler = new SonarModelOutputParser(files);

        depths = new double[files.length];
        int i = 0;

        while (i < files.length) {

            blade = new DepthBlade();
            ranges = new double[numRanges];
```
150

```java
         excess = new double [numRadials][numRanges];
         radialIndex = 0;
         seIndex = 0;
         rangeIndex = 0;

         SAXParserFactory factory = SAXParserFactory.newInstance();

         try {
            // Set up output stream
            out = new OutputStreamWriter(System.out, "UTF8");

            // Parse the input
            SAXParser saxParser = factory.newSAXParser();
            System.out.println("Parsing " + files[i] + "...");
            saxParser.parse( new File(files[i]), handler);

         } catch (Throwable t) {
            t.printStackTrace();
         }

         blade.setBladeParameters(depths[depthIndex-1], ranges, excess, numRadials);

         depthBlades.add(blade);
         i++;
      }

      blades[DEPTH] = depthBlades;
      blades[RADIAL] = generateRadialBlades(depthBlades, numRadials);

      return blades;
   }// end parseDocuments

   /** method creates RadialBlades from parsed data */
   private LinkedList generateRadialBlades(LinkedList depthBlades, int radials) {

      temp = (LinkedList)depthBlades.clone();
      LinkedList radialList = new LinkedList();

      int size = temp.size();

      DepthBlade [] bladeArray = new DepthBlade[size];
      int i = 0;
```

151

```java
    while (i < size) {
      bladeArray[i] = (DepthBlade)temp.removeFirst();
      i++;
    }

    for(int j = 0; j < radials; j++) {

      double[][] slice = new double[size][numRanges];

      i = 0;

      while (i < size) {

        double[] tempArray = bladeArray[i].getRadial(j);

        int k = 0;

        while (k < numRanges) {

          slice[i][k] = tempArray[k];
          k++;
        }// end k while
        i++;
      }// end i while

      RadialBlade b = new RadialBlade(slice, depths, ranges);
      radialList.add(b);
    }// end for

    return radialList;
  }// end generateRadialBlades

  /** converts sound pressure ratio to signal excess */
  public double convertToSignalExcess(double ratio) {

    double se = 20*Math.log(ratio)/Math.log(10);

    if(se < -100) {
      se = -100;
    }

    return se;
  }// end converttoSignalExcess
```

152

```java
   public void setEnvironmentalData (EnvironmentalData data) {

      this.data = data;

      liveData = true;

   } // end setEnvironmentalData


   public void generateAllSonarPlots() {

      int i = 0;
      int counter = 0;

      double [] bearings = new double[] { 0, 45, 90, 135, 180, 225, 270, 315 };

      System.out.println("DEPTHS LENGTH IS " + depths.length);
      while(i < depths.length) {

         // bullseye
         generateBullsEyePlot(Math.rint(depths[i]*KM_TO_FEET));

         // slob
         int j = 0;
         while (j < bearings.length) {

            //generateSLOBChart(bearings[j], depths[i], counter++);
            j++;
         }

         i++;
      }

      generateStackScenePlot();
      generateIsoSurfacePlot();
   }// end generateAllSonarPlots

//**********************************************************************************

   public String generateBullsEyePlot(double depth) {

      LinkedList[] lists = new LinkedList[2];

      //lists = parseDocuments(files);
      lists = (LinkedList[])blades.clone();
```

153

```java
    LinkedList list1 = lists[DEPTH];

    LinkedList temp = (LinkedList)list1.clone();

    int dIndex = 0;
    double kmDepth = depth*FEET_TO_KM;

    int i = 0;

    while(i < depths.length) {
      if(kmDepth >= depths[i]){
        dIndex = i;
      }
      i++;
    }

    DepthBlade blade0 = (DepthBlade)temp.remove(dIndex);

    BullsEyePlot bull = new BullsEyePlot(blade0);

    String filename = bull.generateSceneFile();
    bull.x3dToVRML();

    return filename;
  }// end generateBullseyeplot

  /** generates stack scene */
  public String generateStackScenePlot() {

    String[] trim;
    StringBuffer test = new StringBuffer();

    if(files.length > MAX_ENTRY_SIZE) {
      trim = trimToMaxEntrySize(files);
    } else {
      trim = files;
    }

    if(!liveData) {
      data = new EnvironmentalData("No Time Selected", 0.0, 0.0, 0.0, 0.0, 0.0, "0.0 2.5 7.5 12.5 17.5 25.0 32.5 40.0 50.0 62.5
```
75.0 100.0 125.0 150.0 200.0 300.0 400.0 500.0 600.0 700.0 800.0 900.0 1000.0 1100.0 1200.0 1300.0 1400.0 1500.0
1750.0 2000.0 2500.0 3000.0 4000.0 5000.0 ", "1537.3109 1537.3748 1537.4329 1537.4681 1537.5813 1537.6991
1538.1611 1538.9879 1539.6555 1537.2329 1535.6943 1525 1521.6444 1512.0728 1502.0647 1496.1265 1492.6624

154

1489.8591 1488.1123 1486.8322 1486.0486 1485.6652 1484 1485 1486 1487 1487 1487.5421 1489.373 1491.9246
1498.7546 1506.8083 1523.3582 1541.1987 ", "0 50 100 ", "280 280 280 ", bath60, 0.033333);

```
    }


    StackScene3D stack = new StackScene3D();
    stack.parseResults(blades);
    stack.setParameters(data);


    String filename = stack.generateSceneFile();
    stack.x3dToVRML();


    return filename;


  }// end generateStackScenePlot


  /** generates isosurface plot */
  public String generateIsoSurfacePlot() {


    LinkedList list2 = blades[RADIAL];
    LinkedList nodes;


    CoverageArea area = new CoverageArea( list2, depths, ranges);
    nodes = area.getCoverageNodes();


    if(!liveData) {
        data = new EnvironmentalData("No Time Selected", 0.0, 0.0, 0.0, 0.0, 0.0, "0.0 2.5 7.5 12.5 17.5 25.0 32.5 40.0 50.0 62.5
```
75.0 100.0 125.0 150.0 200.0 300.0 400.0 500.0 600.0 700.0 800.0 900.0 1000.0 1100.0 1200.0 1300.0 1400.0 1500.0
1750.0 2000.0 2500.0 3000.0 4000.0 5000.0 ", "1537.3109 1537.3748 1537.4329 1537.4681 1537.5813 1537.6991
1538.1611 1538.9879 1539.6555 1537.2329 1535.6943 1525 1521.6444 1512.0728 1502.0647 1496.1265 1492.6624
1489.8591 1488.1123 1486.8322 1486.0486 1485.6652 1484 1485 1486 1487 1487 1487.5421 1489.373 1491.9246
1498.7546 1506.8083 1523.3582 1541.1987 ", "0 50 100 ", "280 280 280 ", bath60, 0.033333);

```
    }


    IsoSurfaceBlockPlot plot = new IsoSurfaceBlockPlot(data, nodes);


    String filename = plot.generateSceneFile();
    plot.x3dToVRML();


     return filename;
  }// end generateIsoSurfacePlot



  /** generates single line of bearing plot */
```

155

```java
public void generateSLOBChart(double bearing, double depth, int counter) {

    StringBuffer slob = new StringBuffer();

    LinkedList[] lists = new LinkedList[2];
    lists = parseDocuments(files);

    LinkedList list1 = lists[DEPTH];
    LinkedList list2 = lists[RADIAL];
    LinkedList temp = (LinkedList)list1.clone();

    int dIndex = 0;
    double kmDepth = depth*FEET_TO_KM;
    int i = 0;

    if(kmDepth > depths[depths.length-1]) {
        dIndex = depths.length-1;
    }
    else {
        while(i < depths.length) {
            if(kmDepth > depths[i]){
                dIndex++;
            }
            i++;
        }
    }
    DepthBlade blade0 = (DepthBlade)temp.remove(dIndex);

    int rIndex = getRadialIndex(bearing);
    double [] slobEx = new double[ranges.length];
    excess = blade0.getSignalExcess();

    int j = 0;
    while(j < ranges.length) {

        slobEx[j] = excess[rIndex][j];

        j++;
    }

    SLOBChart chart = new SLOBChart(blade0.getRanges(), slobEx, blade0.getBladeDepth(), bladeBearing, counter);
    chart.createSLOBChart();
```

156

```java
  }// end generateSLOBPlot


/** generates Full Field plot */
public String generateFullFieldPlot(double bearing) {

    LinkedList[] lists = new LinkedList[2];

    //lists = parseDocuments(files);
    lists = (LinkedList[])blades.clone();

    LinkedList list1 = lists[RADIAL];

    LinkedList temp = (LinkedList)list1.clone();


    int i = 0;

    int rIndex = getRadialIndex(bearing);

    RadialBlade blade0 = (RadialBlade)temp.remove(rIndex);

    LineOfBearingPlot full = new LineOfBearingPlot(blade0, bladeBearing);

    String filename = full.generateSceneFile();
    full.x3dToVRML();

    return filename;
  }// end generateFullFieldPlot


//*******************************************************************************

  /** To keep visualization managable, number of BullsEye plots are limited to
   *  amount specified in MAX_ENTRY_SIZE */
  public String[] trimToMaxEntrySize(String[] files) {

    String[] trimmedFiles = new String[MAX_ENTRY_SIZE];

    int i = 0;

    while(i<MAX_ENTRY_SIZE) {

      trimmedFiles[i] = files[i];
```

157

```java
      i++;

   }


   return trimmedFiles;

}// end trimToMaxEntrySize


/** generates radial indices -  assumes radials are in order */

public int getRadialIndex(double bearing){


   int ind = 0;

   double angleGap = 360/numRadials;

   double tempAngle = -1/2*angleGap;

   int i = 0;


   while(i < numRadials) {


      if(bearing > (angleGap + tempAngle)) {


         tempAngle += angleGap;


         ind++;


      }

      i++;

   }


   if(ind == numRadials) {

      ind--;

   }


   bladeBearing = ind*angleGap;


   return ind;

}// end getRadialIndex


public LinkedList[] getBlades() {


   return blades;

}// end getBlades


//********************************************************************************


/** generates Xj3D-friendly version  */
```

```java
public String generateVRMLBullPlot(double depth) {

    StringBuffer test = new StringBuffer();

    LinkedList[] lists = new LinkedList[2];

    lists = parseDocuments(files);

    LinkedList list1 = lists[0];
    LinkedList list2 = lists[1];

    LinkedList temp = (LinkedList)list1.clone();

    int dIndex = 0;
    double kmDepth = depth*FEET_TO_KM;

    int i = 0;

    while(i < depths.length) {
      if(kmDepth > depths[i]){
         dIndex = i;

       }

      i++;
    }
    DepthBlade blade0 = (DepthBlade)temp.remove(dIndex);

    BullsEyePlot bull = new BullsEyePlot(blade0);

    test.append(bull.createVRML(true));

    return test.toString();
  }// end generateVRMLBullseyePlot

/** temporary class generates Xj3D-friendly version single line of bearing plot */
  public String getVRMLSLOBChart(double bearing, double depth, int counter) {

    StringBuffer slob = new StringBuffer();

    LinkedList[] lists = new LinkedList[2];

    lists = parseDocuments(files);
```

159

```java
        LinkedList list1 = lists[0];
        LinkedList list2 = lists[1];

        LinkedList temp = (LinkedList)list1.clone();

        int dIndex = 0;
        double kmDepth = depth*FEET_TO_KM;

        int i = 0;

        if(kmDepth > depths[depths.length-1]) {
            dIndex = depths.length-1;
        }
        else {
            while(i < depths.length) {
                if(kmDepth > depths[i]){
                    dIndex++;
                }
                i++;
            }
        }
        DepthBlade blade0 = (DepthBlade)temp.remove(dIndex);

        int rIndex = getRadialIndex(bearing);
        double [] slobEx = new double[ranges.length];
        excess = blade0.getSignalExcess();

        int j = 0;
        while(j < ranges.length) {

            slobEx[j] = excess[rIndex][j];

            j++;
        }
        SLOBChart chart = new SLOBChart(blade0.getRanges(), slobEx, blade0.getBladeDepth(), bladeBearing, counter);

        return slob.toString();
    }// end getVRMLSLOBPlot
}// end SonarModelOutputParser
```

APPENDIX G. DEPTHBLADE.JAVA

```
/* _____

    Title:        DepthBlade.java

    Description:  Holds sonar visualization data defined by a specified depth. Includes an

                  array of signal excess points by radial and ranges

    Created:      May 20, 2004, 9:33 AM

    Revised:      14 June 2004

    Project:      Tactical Web Services - Sonar Visualization

                  Naval Postgraduate School, Monterey, CA

    Compiler:     JDK 1.4.2

    Author:       Scott Rosetti, LT USN

    Version:      1.0


    _____

*/

package nps.navy.mil.xmsf.SonarVis;

/**

 * This class holds sonar visualization data defined by a specified depth. Includes an

 * array of signal excess points by radial and ranges

 */

public class DepthBlade {

  /** array of ranges of data points from source in KM */
  private double[] ranges;


  /** depth of this depth blade in KM */
  private double depth;


  /** array of signal excess values by radial and range */
  private double[][] excess;


  /** number of radials for this depth */
  private int numRadials;


  /** distance between data points along specified radial */
  private double spacing;


  /** Creates a new instance of DepthBlade */
  public DepthBlade() {
  }
```

```java
/** sets blade parameters */
public void setBladeParameters(double depth, double[] ranges, double[][]excess, int numRadials) {

    this.depth = depth;
    this.ranges = ranges;
    this.excess = excess;
    this.numRadials = numRadials;
    spacing = (ranges[2]-ranges[1]);
}

/** set range values (in KM) */
public void setRanges(double[] values) {
    ranges = values;
}

/** set current blade depth (in KM) */
public void setBladeDepth(double value) {
    depth = value;
}

/** returns array of signal excess values by radial and range */
public void setSignalExcess(double[][] excess) {
    this.excess = excess;
}

/** returns array of depths (in KM) */
public double[] getRanges() {
    return ranges;
}

/** returns range fartest from source */
public double getGreatestRange() {
    return ranges[ranges.length-1];
}

/** returns current blade depth (in KM) */
public double getBladeDepth() {
    return depth;
}

/** returns number of radials */
public int getNumberRadials() {
    return numRadials;
```

162

```java
    }


    /** returns array of signal excess values by radial and range */
    public double [][] getSignalExcess() {
        return excess;
    }


    /** returns array of signal excess values for a specified radial */
    public double[] getRadial(int index) {
        double[] radial = new double[ranges.length];
        int i = 0;

        while (i < ranges.length) {
            radial[i] = excess[index][i];
            i++;
        }
        return radial;
    }


    /** returns current spacing */
    public double getSpacingInKM() {
        return spacing;
    }
}// end DepthBlade
```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX H. RADIALBLADE.JAVA

```
/* _____

    Title:          RadialBlade.java

    Description:     Holds sonar visualization data defined by a specified radial. Includes an

                     array of signal excess points by depth and ranges

    Created:         May 20, 2004, 10:12 AM

    Revised:         14 June 2004

    Project:         Tactical Web Services - Sonar Visualization

                     Naval Postgraduate School, Monterey, CA

    Compiler:        JDK 1.4.2

    Author:          Scott Rosetti, LT USN

    Version:         1.0

   _____

*/


package nps.navy.mil.xmsf.SonarVis;


/**
 *
 * This class holds sonar visualization data defined by a specified radial. Includes an
 * array of signal excess points by depth and ranges
 */
public class RadialBlade {

  /** array of signal excess values per depth and range */
  private double[][] excess;

  /** array of depths in KM */
  private double[] depths;

  /** array of ranges of data points from source in KM*/
  private double[] ranges;

  /** spacing is distance between data points along raidal in KM*/
  private double spacing;

  /** Creates a new instance of RadialBlade (size by ranges) */
  public RadialBlade(double[][] excess, double[] depths, double[] ranges) {
     this.excess = excess;
     this.depths = depths;
```

```java
    this.ranges = ranges;

    spacing = ranges[2]-ranges[1];
  }

  /** return signal excess values */
  public double[][] getSignalExcess() {
    return excess;
  }

  /** return array of blade depths */
  public double[] getDepths() {
    return depths;
  }

  /** returns number of depths of radials*/
  public int getNumDepths() {
   return depths.length;
  }

  /** returns array of ranges of data points from source*/
  public double[] getRanges() {
    return ranges;
  }

  /** returns current spacing */
  public double getSpacingInKM() {
    return spacing;
  }

}// end RadialBlade
```

APPENDIX I. BULLSEYEPLOT.JAVA

```
/* _____

   Title:          BullsEyePlot.java

   Description:    This class generates a bullseye plot with legend in X3D.

   Created:        May 26, 2004, 1:03 PM

   Revised:        14 June 2004

   Project:        Tactical Web Services - Sonar Visualization

                   Naval Postgraduate School, Monterey, CA

   Compiler:       JDK 1.4.2

   Author:         Scott Rosetti, LT USN

   Version:        1.0

   _____

*/

package nps.navy.mil.xmsf.SonarVis;


import java.util.LinkedList;

import java.io.File;

import java.io.Writer;

import java.io.BufferedWriter;

import java.io.FileWriter;

import java.net.URL;

import java.util.Calendar;


import SonarVis.Metcast.MetcastUtilities.XsltUtility;

/**

 * This class generates a BullsEye plot based on valid XML files generated from a sonar model engine.

 */

public class BullsEyePlot {

  /** directory location for native XML files */

  private static final String XML_HOME = "build/MetcastFiles/XML/";


  /** directory location for textured images */

  private static final String IMAGE_HOME = "build/MetcastFiles/Images/";


  /** scene name of generated visualization */

  private static final String SCENE_NAME = "BullsEyePlotExample";


  /** filename of X3D-to-VRML97 XSLT */

  private static final String XSLT = "X3dToVrml97.xslt";
```

```java
/** converts kilometers to feet */
private static final double KM_TO_FEET = 3280.839895013123;

/** converts meters to yards */
private static final double M_TO_YARDS = 1.0936132983377078;

/** converts kilometers to yards */
private static final double KM_TO_YARDS = 1093.6132983377078;

/** converts kilometers to meters (duh) */
private static final double KM_TO_METERS = 1000;

/** first inner ring to be rendered */
private static final int INNER_CUTOFF = 1;

/** objecth that holds signal excess data */
private DepthBlade blade;

/** depth of blade */
private double depth;

/** array of ranges */
private double[] ranges;

/** number of radials */
private int numRadials;

/** number of ranges */
private int numRanges;

/** array of signal excess data */
private double[][] excess;

/** array to hold "rings" of signal lexcess data */
private double[][] rings;

/** full grid signal excess array indexed by numDepths, numRadials, and numRanges */
private double[][][]grid;

/** default half-spacing between ranges */
private double inSpacing = 250;
```

```java
/** default half-spacing between ranges */
private double outSpacing = 250;

/** Creates a new instance of BullsEyePlot */
public BullsEyePlot(DepthBlade blade) {

    this.blade = blade;
    depth = blade.getBladeDepth();
    ranges = blade.getRanges();
    excess = blade.getSignalExcess();
    numRadials = blade.getNumberRadials();
    //spacing = blade.getSpacingInKM()*KM_TO_YARDS/2;

    numRanges = ranges.length;
    rings = new double [numRanges][numRadials];

    grid = createInterpolatedGrid(excess);

}// end constructor



/** Generates properly formatted X3D Stack Scene */
public String generateSceneFile() {

    String path = XML_HOME + SCENE_NAME + "_at depth_" + Math.rint(depth*KM_TO_FEET) + ".x3d";
    File file = new File(path);

    try {

        BufferedWriter writer = new BufferedWriter(new FileWriter(file));
        writer.write(createX3D(true));
        writer.close();
    } catch (Exception e) {
        System.out.println("ERROR: " + e);
    }

    System.out.println("BullsEye Plot Generated");

    return file.getAbsolutePath();

} // end generateSceneFile

/** creates properly formatted X3D subcode for a BullEye plot */
```

```java
public String createX3D(boolean isMultiColored) {

    StringBuffer doc = new StringBuffer();

    if (isMultiColored) {
        doc.append(createX3DHeader());
        doc.append(createMultiColoredScene());

        doc.append("</X3D>");
    } else {
        doc.append(createX3DHeader());
        doc.append(createScene());

        doc.append("</X3D>");
    }
    return doc.toString();
}// end createX3D

/** generates properly formatted X3D header */
private String createX3DHeader() {

    Calendar date = Calendar.getInstance();

    StringBuffer head = new StringBuffer("<?xml version='1.0' encoding='UTF-8'?> \n<!DOCTYPE X3D PUBLIC
    'http://www.web3d.org/specifications/x3d-3.0.dtd' ");
    head.append("'file:///www.web3d.org/TaskGroups/x3d/translation/x3d-3.0.dtd'>\n
    <!--Warning:  transitional DOCTYPE in source .x3d file--> ");

    head.append("\n<X3D profile='Immersive' xmlns:xsd='http://www.w3.org/2001/XMLSchema-instance'
    xsd:noNamespaceSchemaLocation='http://www.web3d.org/specifications/x3d-3.0.xsd'> \n");

    head.append("\n\t<head>\n\t\t<meta content='" + SCENE_NAME + ".x3d' name='filename'/>\n");

    head.append("          \t\t<meta content='Stack Scene sonar visualization auto-generated by Metcast Web Service'
            name='description'/>\n");

    head.append("          \t\t<meta content='" + date.getTime().toString() + "' name='created'/>\n");
    head.append("          \t\t<meta content='" + date.getTime().toString() + "' name='revised'/>\n");
    head.append("          \t\t<meta content='LT Scott Rosetti USN' name='author'/>\n");
    head.append("          \t\t<meta content='BullsEyePlotExample.java' name='generator'/>\n");
    head.append("\t</head>\n");

    return head.toString();
```

170

```java
	}   // end createX3DHeader


/** generates properly formatted subcode for X3D scene (red-yellow-blended version) */
private String createScene() {

    int i = 0;
    StringBuffer scene = new StringBuffer("\t<Scene>\n");
    scene.append(createViewpoints());
    scene.append("\t\t<NavigationInfo type='\"EXAMINE\" \"ANY\"' visibilityLimit=\"200000\" avatarSize=\"0.25 1.6 0.75\"/>
        <!—scene graph nodes are added here -->\n");
    scene.append(createBlendedRangeBlades());
    scene.append(createHUD());
    scene.append(createVisuals());
    scene.append("\t\t<PointLight ambientIntensity='0' intensity='.2' location='0 50000 0' radius='100000'/>\n");
    scene.append("\t</Scene>\n");

    return scene.toString();
}   // end createScene



/** generates properly formatted subcode for X3D scene (multi-colored version) */
private String createMultiColoredScene() {

    int i = 0;

    StringBuffer scene = new StringBuffer("\t<Scene>\n");
    scene.append(createViewpoints());
    scene.append("\t\t<NavigationInfo type='\"EXAMINE\" \"ANY\"'/><!--Scene graph nodes are added here -->\n");
    scene.append(createVisuals());
    scene.append(createMultiColoredSmoothRangeBlades());
    scene.append(createHUD());
    scene.append("\t\t<PointLight ambientIntensity='0' intensity='.2' location='0 50000 0' radius='100000'/>\n");
    scene.append("\t</Scene>\n");

    return scene.toString();
}   // end createMultiColoredScene


/** generates properly formatted subcode for X3D viewpoints */
private String createViewpoints() {

    StringBuffer views = new StringBuffer();

    views.append("\t\t<Viewpoint DEF='Initial_View' description='Initial View' jump='false' orientation='1 0 0 -1.57'
```

```
   position='0 250000 0'/>\n\n");
   views.append("\t\t<Viewpoint DEF='Zoom1_View' description='Zoom 1' jump='false' orientation='1 0 0 -1.57'
   position='0 100000 0'/>\n\n");
   views.append("\t\t<Viewpoint DEF='Zoom2_View' description='Zoom 2' jump='false' orientation='1 0 0 -1.57'
   position='0 50000 0'/>\n\n");
   views.append("\t\t<Viewpoint DEF='Zoom3_View' description='Zoom 3' jump='false' orientation='1 0 0 -1.57'
   position='0 20000 0'/>\n\n");
   views.append("\t\t<Viewpoint DEF='Zoom4_View' description='Zoom 4' jump='false' orientation='1 0 0 -1.57'
   position='0 10000 0'/>\n\n");
   return views.toString();

}    // end createViewpoints

/** generates a grid of blended values that uses adjacent interpolation */
public double[][][] createInterpolatedGrid(double[][] values) {

   double[][][] grid = new double[numRadials][numRanges][9];
   int i = 0;

   while (i < numRadials) {
      int j = 0;
      while(j < numRanges) {

         int jplus = j;
         int jminus = j;
         int iplus = i;
         int iminus = i;

         if (j == 0) {
            jminus = 0;
         } else if (j >= (numRanges-1)) {
            jplus = j;
         } else {
            jplus = j+1;
            jminus = j - 1;
         }

         if (i == 0) {
            iminus = numRadials-1;
         } else if (i >= (numRadials-1)) {
            iplus = 0;
         } else {
            iplus = i+1;
```

172

```java
                iminus = i - 1;
            }


            grid[i][j][0] = values[i][j];
            grid[i][j][1] = values[i][j];
            grid[i][j][2] = values[i][j];
            grid[i][j][3] = values[i][j];
            grid[i][j][4] = values[i][j];
            grid[i][j][5] = values[i][j];
            grid[i][j][6] = values[i][j];
            grid[i][j][7] = values[i][j];
            grid[i][j][8] = values[i][j];


            j++;
        }
        i++;
    }


    return grid;
}    // end createInterpolatedGrid


/** creates blended bullseye plot using interpolated grid */
public String createBlendedRangeBlades() {

    int i = 0;
    StringBuffer rings = new StringBuffer();
    StringBuffer colorIndex = new StringBuffer();
    StringBuffer coordIndex = generateBlendedCoordScheme();
    StringBuffer rangeMarkers = new StringBuffer();

    rings.append("\t<Transform rotation=\"0 1 0 1.57\">");

    while(i < numRadials) { // numRadials
        colorIndex = new StringBuffer();
        rings.append("\t\t<Transform>\n");

        int j = INNER_CUTOFF;

        double angle = 2 * i * Math.PI/numRadials;
        double halfAngle = 2 * Math.PI/(numRadials * 2);
        double thirdAngle = 2 * Math.PI/(numRadials * 6);

        rings.append("\t\t\t<Shape>\n\t\t\t\t<IndexedFaceSet solid=\"false\" coordIndex=\"" + coordIndex.toString() + "\"
```
173

```
colorPerVertex=\"false\">\n\t\t\t\t\t<Coordinate point=\"");

    while (j < numRanges) { //numRanges

        double colorR;
        double colorG;
        double colorB;

        int k = 0;

        if (j>INNER_CUTOFF) {

            while (k < 9) {
                if(grid[i][j][k] >= 25) {
                    colorR = 1;
                    colorG = 0;
                    colorB = 0;
                } else if (grid[i][j][k] > 0) {
                    colorR = 1;
                    colorG = 0.5;
                    colorB = 0;
                } else {
                    colorR = 1;
                    colorG = 1;
                    colorB = 0;
                }

                colorIndex.append(colorR + " " + colorG + " " + colorB + " ");

                k++;
            }

        }
        if(j != 0) {
            // add first row of points
            if(j == 1) {

                inSpacing = (ranges[1]-ranges[0])*KM_TO_YARDS/2;

                rings.append((Math.cos(angle+halfAngle)*(ranges[j]*KM_TO_YARDS-inSpacing)) + " 0 " +
            (Math.sin(angle+halfAngle)*(ranges[j]*KM_TO_YARDS-inSpacing)) + ", ");

                rings.append((Math.cos(angle+thirdAngle)*(ranges[j]*KM_TO_YARDS-inSpacing)) + " 0 " +
```
174

```
(Math.sin(angle+thirdAngle)*(ranges[j]*KM_TO_YARDS-inSpacing)) + ", ");


     rings.append((Math.cos(angle-thirdAngle)*(ranges[j]*KM_TO_YARDS-inSpacing)) + " 0 " + (Math.sin(angle-
thirdAngle)*(ranges[j]*KM_TO_YARDS-inSpacing)) + ", ");


     rings.append((Math.cos(angle-halfAngle)*(ranges[j]*KM_TO_YARDS-inSpacing)) + " 0 " + (Math.sin(angle-
halfAngle)*(ranges[j]*KM_TO_YARDS-inSpacing)) + ", ");
   }


   inSpacing = (ranges[j] - ranges[j-1])*KM_TO_YARDS/2;
   if(j == (numRanges-1)) { outSpacing = 250; }
   else { outSpacing = (ranges[j+1] - ranges[j])*KM_TO_YARDS/2; }


   if(j == (numRanges-1)) { outSpacing = 250; }


   rings.append((Math.cos(angle+halfAngle)*(ranges[j]*KM_TO_YARDS-inSpacing/3)) + " 0 " +
(Math.sin(angle+halfAngle)*(ranges[j]*KM_TO_YARDS-inSpacing/3)) + ", ");


   rings.append((Math.cos(angle+thirdAngle)*(ranges[j]*KM_TO_YARDS-inSpacing/3)) + " 0 " +
(Math.sin(angle+thirdAngle)*(ranges[j]*KM_TO_YARDS-inSpacing/3)) + ", ");


   rings.append((Math.cos(angle-thirdAngle)*(ranges[j]*KM_TO_YARDS-inSpacing/3)) + " 0 " + (Math.sin(angle-
thirdAngle)*(ranges[j]*KM_TO_YARDS-inSpacing/3)) + ", ");


   rings.append((Math.cos(angle-halfAngle)*(ranges[j]*KM_TO_YARDS-inSpacing/3)) + " 0 " + (Math.sin(angle-
halfAngle)*(ranges[j]*KM_TO_YARDS-inSpacing/3)) + ", ");


   rings.append((Math.cos(angle+halfAngle)*(ranges[j]*KM_TO_YARDS+outSpacing/3)) + " 0 " +
(Math.sin(angle+halfAngle)*(ranges[j]*KM_TO_YARDS+outSpacing/3)) + ", ");


   rings.append((Math.cos(angle+thirdAngle)*(ranges[j]*KM_TO_YARDS+outSpacing/3)) + " 0 " +
(Math.sin(angle+thirdAngle)*(ranges[j]*KM_TO_YARDS+outSpacing/3)) + ", ");


   rings.append((Math.cos(angle-thirdAngle)*(ranges[j]*KM_TO_YARDS+outSpacing/3)) + " 0 " + (Math.sin(angle-
thirdAngle)*(ranges[j]*KM_TO_YARDS+outSpacing/3)) + ", ");


   rings.append((Math.cos(angle-halfAngle)*(ranges[j]*KM_TO_YARDS+outSpacing/3)) + " 0 " + (Math.sin(angle-
halfAngle)*(ranges[j]*KM_TO_YARDS+outSpacing/3)) + ", ");


   rings.append((Math.cos(angle+halfAngle)*(ranges[j]*KM_TO_YARDS+outSpacing)) + " 0 " +
(Math.sin(angle+halfAngle)*(ranges[j]*KM_TO_YARDS+outSpacing)) + ", ");


   rings.append((Math.cos(angle+thirdAngle)*(ranges[j]*KM_TO_YARDS+outSpacing)) + " 0 " +
```

```
                (Math.sin(angle+thirdAngle)*(ranges[j]*KM_TO_YARDS+outSpacing)) + ", ");

            rings.append((Math.cos(angle-thirdAngle)*(ranges[j]*KM_TO_YARDS+outSpacing)) + " 0 " + (Math.sin(angle-
        thirdAngle)*(ranges[j]*KM_TO_YARDS+outSpacing)) + ", ");

            rings.append((Math.cos(angle-halfAngle)*(ranges[j]*KM_TO_YARDS+outSpacing)) + " 0 " + (Math.sin(angle-
        halfAngle)*(ranges[j]*KM_TO_YARDS+outSpacing)) + ", ");

        }

        j++;
    }

    rings.append((Math.cos(angle-halfAngle)*(ranges[numRanges-1]*KM_TO_YARDS+outSpacing)) + " 0 " + (Math.sin(angle-
    halfAngle)*(ranges[numRanges-1]*KM_TO_YARDS+outSpacing)) + ", " + (Math.cos(angle+halfAngle)*
    (ranges[numRanges-1]*KM_TO_YARDS+outSpacing)) + " 0 " + (Math.sin(angle+halfAngle)
    *(ranges[numRanges-1]*KM_TO_YARDS+outSpacing)));

    rings.append("\"/>\n\t\t\t\t\t<Color color=\"" + colorIndex.toString() + "\"/>\n\t\t\t\t\t</IndexedFaceSet>\n");

    rings.append("\t\t\t</Shape>\n\t\t</Transform>\n");
    rings.append("\t</Transform>");
    i++;
    }
    return rings.toString();
} // end createBlendedRangeBlades

/** creates smooth multi-colored bullseye plot (3 cell) */
public String createMultiColoredSmoothRangeBlades() {

    int i = 0;
    StringBuffer rings = new StringBuffer();
    StringBuffer colorIndex = new StringBuffer();
    StringBuffer coordIndex = generateSmoothCoordScheme();

    StringBuffer rangeMarkers = new StringBuffer();

    while(i < numRadials) { // numRadials
        colorIndex = new StringBuffer();
        rings.append("\t\t<Transform rotation=\"0 1 0 1.57\">\n");

        int j = INNER_CUTOFF;
```

```java
double angle = 2 * i * Math.PI/numRadials;
double halfAngle = 2 * Math.PI/(numRadials * 2);
double thirdAngle = 2 * Math.PI/(numRadials * 6);


rings.append("\t\t\t<Shape>\n\t\t\t\t<IndexedFaceSet solid=\"false\" coordIndex=\"" + coordIndex.toString() + "\"
    colorPerVertex=\"false\">\n\t\t\t\t\t<Coordinate point=\"");

while (j < numRanges) { //numRanges

  double colorR;
  double colorG;
  double colorB;

  int k = 0;

  if (j>INNER_CUTOFF) {
    while (k < 3) {
      if(grid[i][j][k] >= 30) {
        colorR = 1; // red
        colorG = 0;
        colorB = 0;
      } else if (grid[i][j][k] >= 25) {
        colorR = 0.949; // orange
        colorG = 0.688;
        colorB = 0.027;
      } else if (grid[i][j][k] >= 20) {
        colorR = 0.949; // dark yellow
        colorG = 0.859;
        colorB = 0.027;
      } else if (grid[i][j][k] >= 15) {
        colorR = 0.935; // yellow
        colorG = 0.910;
        colorB = 0.129;
      } else if (grid[i][j][k] >= 10) {
        colorR = 0.527; // light green
        colorG = 0.867;
        colorB = 0.116;
      } else if (grid[i][j][k] >= 5) {
        colorR = 0.277; // dark green
        colorG = 0.793;
        colorB = 0.191;
      } else if (grid[i][j][k] >= 0) {
        colorR = 0.134; // skyblue
```

```
              colorG = 0.776;
              colorB = 0.847;
          } else if (grid[i][j][k] >= -5) {
              colorR = 0.352; // light blue
              colorG = 0.605;
              colorB = 0.618;
          } else if (grid[i][j][k] >= -10) {
              colorR = 0.012; // medium blue
              colorG = 0.129;
              colorB = 0.956;
          } else {
              colorR = 0.449; // dark blue
              colorG = 0.018;
              colorB = 0.953;
          }
          colorIndex.append(colorR + " " + colorG + " " + colorB + " ");
          k++;
      }


  }
  if(j >= INNER_CUTOFF) {
      inSpacing = (ranges[j] - ranges[j-1])*KM_TO_YARDS/2;
      if(j == (numRanges-1)) { outSpacing = 250; }
      else { outSpacing = (ranges[j+1] - ranges[j])*KM_TO_YARDS/2; }


      if (j == (numRanges-1)) { outSpacing = 250; }


      rings.append((Math.cos(angle+halfAngle)*(ranges[j]*KM_TO_YARDS+outSpacing)) + " 0 " +
  (Math.sin(angle+halfAngle)*(ranges[j]*KM_TO_YARDS+outSpacing)) + ", ");


      rings.append((Math.cos(angle+thirdAngle)*(ranges[j]*KM_TO_YARDS+outSpacing)) + " 0 " +
  (Math.sin(angle+thirdAngle)*(ranges[j]*KM_TO_YARDS+outSpacing)) + ", ");


      rings.append((Math.cos(angle-thirdAngle)*(ranges[j]*KM_TO_YARDS+outSpacing)) + " 0 " + (Math.sin(angle-
  thirdAngle)*(ranges[j]*KM_TO_YARDS+outSpacing)) + ", ");


      rings.append((Math.cos(angle-halfAngle)*(ranges[j]*KM_TO_YARDS+outSpacing)) + " 0 " + (Math.sin(angle-
  halfAngle)*(ranges[j]*KM_TO_YARDS+outSpacing)) + ", ");
  }
  j++;
}

rings.append((Math.cos(angle-halfAngle)*(ranges[numRanges-1]*KM_TO_YARDS+outSpacing)) + " 0 " + (Math.sin(angle-
```

```
            halfAngle)*(ranges[numRanges-1]*KM_TO_YARDS+outSpacing)) + ", " +
            (Math.cos(angle+halfAngle)*(ranges[numRanges-1]*KM_TO_YARDS+outSpacing)) + " 0 " +
            (Math.sin(angle+halfAngle)*(ranges[numRanges-1]*KM_TO_YARDS+outSpacing)));


    rings.append("\"/>\n\t\t\t\t\t<Color color=\"" + colorIndex.toString() + "\"/>\n\t\t\t\t</IndexedFaceSet>\n");
    rings.append("\t\t\t</Shape>\n\t\t</Transform>\n");
    i++;
  }



  return rings.toString();
} // end createMultiColoredSmoothRangeBlades


/** creates coordinat scheme for basic plot */
private StringBuffer generateCoordScheme() {

  StringBuffer scheme = new StringBuffer();


  int j = 0;
  while (j < (numRanges - INNER_CUTOFF)*2) {
    scheme.append(j + " " + (j+1) + " " + (j+3) + " " + (j+2) + " " + j + " -1 ");
    j+=2;
  }


  return scheme;
}// end generateCoordscheme


/** creates coordinate scheme for blended plot */
private StringBuffer generateBlendedCoordScheme() {

  StringBuffer scheme = new StringBuffer();
  int j = 0;
  while (j < (numRanges - INNER_CUTOFF-1)*12) {
    scheme.append(j + " " + (j+1) + " " + (j+5) + " " +  (j+4) + " " + j + " -1 ");
    j++;
    scheme.append(j + " " + (j+1) + " " + (j+5) + " " +  (j+4) + " " + j + " -1 ");
    j++;
    scheme.append(j + " " + (j+1) + " " + (j+5) + " " +  (j+4) + " " + j +" -1 ");
    j+=2;
  }


  return scheme;
}// end generatedBlendedCOordScheme
```

179

```java
/** creates coordinate scheme for smooth plot (3 cell per data point) */
private StringBuffer generateSmoothCoordScheme() {

    StringBuffer scheme = new StringBuffer();

    int j = 0;

    while (j < ((numRanges - INNER_CUTOFF-1)*4)) {
        scheme.append(j + " " + (j+1) + " " + (j+5) + " " +  (j+4) + " " + j + " -1 ");
        j++;
        scheme.append(j + " " + (j+1) + " " + (j+5) + " " +  (j+4) + " " + j + " -1 ");
        j++;
        scheme.append(j + " " + (j+1) + " " + (j+5) + " " +  (j+4) + " " + j +" -1 ");
        j+=2;
    }

    return scheme;
}// end generateSmoothCoordSCheme



/** creates heads up display - UNFINISHED */
private String createHUD() {

    StringBuffer hud = new StringBuffer();

    //add HUD
    hud.append("\t\t<Switch whichChoice='0'>\n\t\t\t<Transform>\n\t\t\t\t<Transform>\n\t\t\t\t\t<ProximitySensor DEF='Viewpoint'
        size='1000000000 1000000000 1000000000'/>\n");
    hud.append("\t\t\t\t\t<Switch whichChoice='0'>\n\t\t\t\t\t\t<Transform DEF='HUD'>\n\t\t\t\t\t\t\t<Transform DEF='dBPanel'
        scale='1 1 1' translation='1.4 -0.25 -3'>\n");

    //legend
    String path = IMAGE_HOME + "legend.jpg";
    File file = new File(path);
    try {
        URL url = file.toURL();
        path = url.toString();

    }catch(Exception e) {
    }

    hud.append("\t\t\t\t<Shape>\n");
```

```java
hud.append("\t\t\t\t\t<IndexedFaceSet coordIndex=\"0 1 2 3 0\">\n");
hud.append("\t\t\t\t\t<Coordinate point=\".3 .5 0, -.3 .5 0, -.3 -.5 0, .3 -.5 0\"/>\n");
hud.append("\t\t\t\t\t<TextureCoordinate point=\"1 1, 0 1, 0 0, 1 0\"/>\n");
hud.append("\t\t\t\t</IndexedFaceSet>\n");
hud.append("\t\t\t\t<Appearance>\n");
hud.append("\t\t\t\t\t<ImageTexture repeatS=\"false\" repeatT=\"false\" url=\"" + path + "\"/>\n");
hud.append("\t\t\t\t</Appearance>\n");
hud.append("\t\t\t</Shape>\n");


hud.append("\t\t\t\t\t\t</Transform>\n");
hud.append("\t\t\t\t\t\t<ROUTE fromField='position_changed' fromNode='Viewpoint' toField='set_translation'
        toNode='HUD'/>\n");
hud.append("\t\t\t\t\t\t<ROUTE fromField='orientation_changed' fromNode='Viewpoint' toField='set_rotation'
        toNode='HUD'/>\n");
hud.append("\t\t\t\t\t</Transform>\n\t\t\t\t\t</Switch>\n\t\t\t\t</Transform>\n\t\t\t</Transform>\n\t\t</Switch>\n");


    return hud.toString();
}// end createHUD

public String createVisuals() {

    StringBuffer vis = new StringBuffer();
    vis.append("\t\t<Transform translation=\"0 100 0\">");


    // n/s
    vis.append("\t\t<Transform rotation=\"0 0 1 1.57\">\n");
    vis.append("\t\t\t<Transform translation=\"0 100 0\">\n");
    vis.append("\t\t\t\t<Shape>\n");
    vis.append("\t\t\t\t\t<Appearance>\n");
    vis.append("\t\t\t\t\t\t<Material diffuseColor=\"0.6 0.6 0\"/>");
    vis.append("\t\t\t\t\t</Appearance>\n");
    vis.append("\t\t\t\t\t<Cylinder height=\"" + (ranges[ranges.length-1]*KM_TO_YARDS*2) + "\" radius=\"200\"/>\n");
    vis.append("\t\t\t\t</Shape>\n");
    vis.append("\t\t\t</Transform>\n");
    vis.append("\t\t</Transform>\n");


    vis.append("\t\t<Transform rotation=\"1 0 0 -1.57\" scale=\"10000 10000 10000\">\n");
    vis.append("\t\t\t<Transform translation=\"-0.5 " + ((ranges[ranges.length-1]*KM_TO_YARDS+5000)/10000) + " 0\">\n");
    vis.append("\t\t\t\t<Shape>\n");
    vis.append("\t\t\t\t\t<Appearance>\n");
    vis.append("\t\t\t\t\t\t<Material diffuseColor=\"0.6 0.6 0\"/>");
    vis.append("\t\t\t\t\t</Appearance>\n");
    vis.append("\t\t\t\t\t<Text string=\"000\"/>\n");
```

181

```
vis.append("\t\t\t\t</Shape>\n");
vis.append("\t\t\t</Transform>\n");
vis.append("\t\t</Transform>\n");


vis.append("\t\t<Transform rotation=\"1 0 0 -1.57\" scale=\"10000 10000 10000\">\n");
vis.append("\t\t\t<Transform translation=\"-0.5 " + (-1*(ranges[ranges.length-1]*KM_TO_YARDS+8000)/10000) + " 0\">\n");
vis.append("\t\t\t\t<Shape>\n");
vis.append("\t\t\t\t\t<Appearance>\n");
vis.append("\t\t\t\t\t\t<Material diffuseColor=\"0.6 0.6 0\"/>");
vis.append("\t\t\t\t\t</Appearance>\n");
vis.append("\t\t\t\t\t<Text string=\"180\"/>\n");
vis.append("\t\t\t\t</Shape>\n");
vis.append("\t\t\t</Transform>\n");
vis.append("\t\t</Transform>\n");


// e/w
vis.append("\t\t<Transform rotation=\"1 0 0 1.57\">\n");
vis.append("\t\t\t<Transform translation=\"0 100 0\">\n");
vis.append("\t\t\t\t<Shape>\n");
vis.append("\t\t\t\t\t<Appearance>\n");
vis.append("\t\t\t\t\t\t<Material diffuseColor=\"0.6 0.6 0\"/>");
vis.append("\t\t\t\t\t</Appearance>\n");
vis.append("\t\t\t\t\t<Cylinder height=\"" + (ranges[ranges.length-1]*KM_TO_YARDS*2) + "\" radius=\"200\"/>\n");
vis.append("\t\t\t\t</Shape>\n");
vis.append("\t\t\t</Transform>\n");
vis.append("\t\t</Transform>\n");


vis.append("\t\t\t<Transform rotation=\"1 0 0 -1.57\" scale=\"10000 10000 10000\">\n");
vis.append("\t\t<Transform translation=\"" + ((ranges[ranges.length-1]*KM_TO_YARDS+5000)/10000) + " 0 0\">\n");
vis.append("\t\t\t\t<Shape>\n");
vis.append("\t\t\t\t\t<Appearance>\n");
vis.append("\t\t\t\t\t\t<Material diffuseColor=\"0.6 0.6 0\"/>");
vis.append("\t\t\t\t\t</Appearance>\n");
vis.append("\t\t\t\t\t<Text string=\"090\"/>\n");
vis.append("\t\t\t\t</Shape>\n");
vis.append("\t\t\t</Transform>\n");
vis.append("\t\t</Transform>\n");


vis.append("\t\t\t<Transform rotation=\"1 0 0 -1.57\" scale=\"10000 10000 10000\">\n");
vis.append("\t\t<Transform translation=\"" + ((ranges[ranges.length-1]*KM_TO_YARDS*(-1)-15000)/10000) + " 0 0\">\n");
vis.append("\t\t\t\t<Shape>\n");
vis.append("\t\t\t\t\t<Appearance>\n");
vis.append("\t\t\t\t\t\t<Material diffuseColor=\"0.6 0.6 0\"/>");
```

```
vis.append("\t\t\t\t\t</Appearance>\n");
vis.append("\t\t\t\t\t<Text string=\"270\"/>\n");
vis.append("\t\t\t\t</Shape>\n");
vis.append("\t\t\t</Transform>\n");
vis.append("\t\t</Transform>\n");


//nw/se
vis.append("\t\t<Transform rotation=\"0 0 1 1.57\">\n");
vis.append("\t\t\t<Transform rotation=\"1 0 0 0.7535\">\n");
vis.append("\t\t\t<Transform translation=\"0 100 0\">\n");
vis.append("\t\t\t\t<Shape>\n");
vis.append("\t\t\t\t\t<Appearance>\n");
vis.append("\t\t\t\t\t\t<Material diffuseColor=\"0.6 0.6 0\"/>");
vis.append("\t\t\t\t\t</Appearance>\n");
vis.append("\t\t\t\t\t<Cylinder height=\"" + (ranges[ranges.length-1]*KM_TO_YARDS*2) + "\" radius=\"200\"/>\n");
vis.append("\t\t\t\t</Shape>\n");
vis.append("\t\t\t</Transform>\n");
vis.append("\t\t\t</Transform>\n");
vis.append("\t\t</Transform>\n");


// sw/ne
vis.append("\t\t<Transform rotation=\"0 0 1 1.57\">\n");
vis.append("\t\t\t<Transform rotation=\"1 0 0 -0.7535\">\n");
vis.append("\t\t\t<Transform translation=\"0 100 0\">\n");
vis.append("\t\t\t\t<Shape>\n");
vis.append("\t\t\t\t\t<Appearance>\n");
vis.append("\t\t\t\t\t\t<Material diffuseColor=\"0.6 0.6 0\"/>");
vis.append("\t\t\t\t\t</Appearance>\n");
vis.append("\t\t\t\t\t<Cylinder height=\"" + (ranges[ranges.length-1]*KM_TO_YARDS*2) + "\" radius=\"200\"/>\n");
vis.append("\t\t\t\t</Shape>\n");
vis.append("\t\t\t</Transform>\n");
vis.append("\t\t\t</Transform>\n");
vis.append("\t\t</Transform>\n");


vis.append("\t\t</Transform>\n");


// range rings
vis.append("\t\t<Transform>\n");


double distance = 0;
while (distance <= (ranges[ranges.length-1]*KM_TO_YARDS)) {


    StringBuffer circle = new StringBuffer();
```

183

```java
        int x = 0;
        int steps = 36;

        while (x <= steps) {

            circle.append(distance*Math.cos(2*x*Math.PI/steps) + " 1000 " + distance*Math.sin(2*x*Math.PI/steps) + ", ");


            x++;
        }



        vis.append("\t\t\t\t<Shape>\n");
        vis.append("\t\t\t\t<Appearance>\n");
        vis.append("\t\t\t\t\t<Material diffuseColor=\".6 .6 0\"/>\n");
        vis.append("\t\t\t\t</Appearance>\n");
        vis.append("\t\t\t\t<Extrusion crossSection=\"100 100, -100 100, -100 -100, 100 -100\" solid=\"false\" spine=\"" +
                    circle.toString() + "\"/>\n");
        vis.append("\t\t\t</Shape>\n");

        distance += 10000;
    }


    vis.append("\t\t</Transform>\n");
    return vis.toString();
}// end createVisuals

/** convers X3D to VRML for browser display using X3dToVrml.xslt*/
public void x3dToVRML() {

    XsltUtility util = new XsltUtility();

    util.runXslt(XML_HOME + SCENE_NAME + "_at depth_" + Math.rint(depth*KM_TO_FEET) + ".x3d", XML_HOME +
            SCENE_NAME + "_at depth_" + Math.rint(depth*KM_TO_FEET) + ".wrl", XML_HOME + XSLT);

    System.out.println(SCENE_NAME + "_at depth_" + Math.rint(depth*KM_TO_FEET) + ".wrl created");
}


//********************* VRML VERSION UNTIL Xj3D WORKS CORRECTLY ******************************

public String createVRML(boolean isMultiColored) {
```

184

```java
        StringBuffer doc = new StringBuffer();

        if (isMultiColored) {
            doc.append(createVRMLHeader());
            doc.append(createVRMLViewpoints());
            doc.append(createVRMLVisuals());
            doc.append(createMultiColoredVRMLRangeRings());


        } else {
            doc.append(createVRMLHeader());
            doc.append(createVRMLViewpoints());
        }
        return doc.toString();
    }

    private String createVRMLHeader() {

        StringBuffer header = new StringBuffer();

        header.append("#X3D V3.0 utf8\n PROFILE Immersive \n");
        header.append("DirectionalLight { direction 0 -1 0 } \n ");
        header.append("NavigationInfo { type [\"FLY\",\"ANY\"] visibilityLimit 300000 speed 250 avatarSize [4 1.5 0.75] }\n");

        // background
        header.append("Background { groundAngle [1.309, 1.571] groundColor [0.1 0.1 0, 0.4 0.25 0.2, 0.6 0.6 0.6] skyAngle [1.309,
            1.571] skyColor [0 0.2 0.7, 0 0.5 1, 1 1 1]}\n");

        return header.toString();
    }

    private String createVRMLViewpoints() {

        StringBuffer views = new StringBuffer();

        views.append("Viewpoint { description \"Original\"  position 0.0 250000 0 orientation 1 0 0 -1.57 }\n");
        views.append("Viewpoint { description \"Zoom 1\"  position 0.0 100000 0 orientation 1 0 0 -1.57 }\n");
        views.append("Viewpoint { description \"Zoom 2\"  position 0.0 50000 0 orientation 1 0 0 -1.57 }\n");
        views.append("Viewpoint { description \"Zoom 3\"  position 0.0 20000 0 orientation 1 0 0 -1.57 }\n");
        views.append("Viewpoint { description \"Zoom 4\"  position 0.0 10000 0 orientation 1 0 0 -1.57 }\n");
        views.append("Viewpoint { description \"Legend\"  position 0 0 1000 }\n");
        views.append("Viewpoint { description \"Legend2\"  position 0 10000 0 orientation 1 0 0 -1.57 }\n");
```

185

```java
        return views.toString();

    }


    public String createVRMLVisuals() {

        StringBuffer vis = new StringBuffer();


        // n/s
        vis.append("Transform {");
        vis.append("   translation 0 20000 0");
        vis.append("   children [");
        vis.append("      Transform {");
        vis.append("         rotation 1 0 0 1.57");
        vis.append("         children [");
        vis.append("            Shape {");
        vis.append("               appearance Appearance{");
        vis.append("                  material Material{ ");
        vis.append("                     diffuseColor 0.6 0.6 0  }}");
        vis.append("               geometry Cylinder {");
        vis.append("                  height " + (ranges[ranges.length-1]*KM_TO_YARDS*2));
        vis.append("                  radius 200  } } ]}  ]}");


        vis.append("Transform { ");
        vis.append("   translation 0 20000 " + (ranges[ranges.length-1]*KM_TO_YARDS*(-1)-5000));
        vis.append("   children [ ");
        vis.append("   Transform { ");
        vis.append("      rotation 1 0 0 -1.57");
        vis.append("      scale 10000 10000 10000 ");
        vis.append("       children [ ");
        vis.append("        Shape { ");
        vis.append("         appearance Appearance{");
        vis.append("           material Material{ ");
        vis.append("              diffuseColor 0.6 0.6 0 }} ");
        vis.append("         geometry Text { ");
        vis.append("            string \"000\"       } } ]}  ]}");
        vis.append("Transform { ");
        vis.append("   translation 0 20000 " + (ranges[ranges.length-1]*KM_TO_YARDS+5000));
        vis.append("   children [ ");
        vis.append("   Transform { ");
        vis.append("      rotation 1 0 0 -1.57");
        vis.append("      scale 10000 10000 10000 ");
        vis.append("       children [ ");
        vis.append("        Shape { ");
```

```
vis.append("          appearance Appearance{");
vis.append("            material Material{ ");
vis.append("             diffuseColor 0.6 0.6 0 }} ");
vis.append("          geometry Text { ");
vis.append("            string \"180\"        }   } ]}   ]}");
vis.append("Transform { ");
vis.append("   translation 75000 20000 " + (ranges[ranges.length-1]*KM_TO_YARDS*(-1)-12500));
vis.append("   children [ ");
vis.append("   Transform { ");
vis.append("     rotation 1 0 0 " + -1*Math.PI/2);
vis.append("     scale 10000 10000 10000 ");
vis.append("      children [ ");
vis.append("       Shape { ");
vis.append("         appearance Appearance{");
vis.append("           material Material{ ");
vis.append("             diffuseColor 0.6 0.6 0 }} ");
vis.append("         geometry Text { ");
vis.append("             string \"" + new Float(Math.round(depth*KM_TO_FEET)).intValue() + "\"        }   } ]}   ]}");


// e/w
vis.append("Transform {");
vis.append("   rotation 0 1 0 1.57");
vis.append("   children [");
vis.append("     Transform {");
vis.append("        translation 0 20000 0");
vis.append("        children [");
vis.append("          Transform {");
vis.append("          rotation 1 0 0 1.57");
vis.append("          children [");
vis.append("           Shape {");
vis.append("              appearance Appearance{");
vis.append("                material Material{ ");
vis.append("                  diffuseColor 0.6 0.6 0  }}");
vis.append("              geometry Cylinder {");
vis.append("                 height " + (ranges[ranges.length-1]*KM_TO_YARDS*2));
vis.append("                 radius 200  } } ]}   ] }   ]}");

vis.append("Transform { ");
vis.append("   translation " + (ranges[ranges.length-1]*KM_TO_YARDS+5000) + " 20000 0");
vis.append("   children [ ");
vis.append("   Transform { ");
vis.append("     rotation 1 0 0 -1.57");
```

187

```
vis.append("    scale 10000 10000 10000 ");
vis.append("      children [ ");
vis.append("       Shape { ");
vis.append("        appearance Appearance{");
vis.append("          material Material{ ");
vis.append("            diffuseColor 0.6 0.6 0 }} ");
vis.append("        geometry Text { ");
vis.append("          string \"090\"        }  } ]}  ]}");
vis.append("Transform { ");
vis.append("   translation " + (ranges[ranges.length-1]*KM_TO_YARDS*(-1)-5000) + " 20000 0");
vis.append("   children [ ");
vis.append("   Transform { ");
vis.append("    rotation 1 0 0 -1.57");
vis.append("    scale 10000 10000 10000 ");
vis.append("      children [ ");
vis.append("       Shape { ");
vis.append("        appearance Appearance{");
vis.append("          material Material{ ");
vis.append("            diffuseColor 0.6 0.6 0 }} ");
vis.append("        geometry Text { ");
vis.append("          string \"270\"        }  } ]}  ]}");

//nw/se
vis.append("Transform {");
vis.append("   rotation 0 1 0 0.7535");
vis.append("   children [");
vis.append("     Transform {");
vis.append("       translation 0 20000 0");
vis.append("       children [");
vis.append("        Transform {");
vis.append("         rotation 1 0 0 1.57");
vis.append("          children [");
vis.append("           Shape {");
vis.append("            appearance Appearance{");
vis.append("             material Material{ ");
vis.append("              diffuseColor 0.6 0.6 0  }}");
vis.append("            geometry Cylinder {");
vis.append("             height " + (ranges[ranges.length-1]*KM_TO_YARDS*2));
vis.append("              radius 200  } } ]}  ] }  ]}");

// sw/ne
vis.append("Transform {");
vis.append("   rotation 0 1 0 -0.7535");
```
188

```java
        vis.append("   children [");
        vis.append("     Transform {");
        vis.append("       translation 0 20000 0");
        vis.append("       children [");
        vis.append("         Transform {");
        vis.append("           rotation 1 0 0 1.57");
        vis.append("           children [");
        vis.append("             Shape {");
        vis.append("               appearance Appearance{");
        vis.append("                 material Material{ ");
        vis.append("                   diffuseColor 0.6 0.6 0  }}");
        vis.append("               geometry Cylinder {");
        vis.append("                 height " + (ranges[ranges.length-1]*KM_TO_YARDS*2));
        vis.append("                 radius 200  } } ]}   ] }   ]}");


        //legend
        String path = "build/MetcastFiles/Images/legend.jpg";
        File file = new File(path);
        try {
            URL url = file.toURL();
            path = url.toString();


        }catch(Exception e) {
        }


        vis.append("Transform { translation 0 2000 0 children [ Transform { rotation 1 0 0 -1.57   scale 80 80 80 translation 1150 244500
            600  children [ Shape { geometry IndexedFaceSet {  coordIndex [ 0 1 2 3 0] coord Coordinate { point [ 3 5 0, -3 5 0, -3 -5
            0, 3 -5 0] } texCoord TextureCoordinate { point [1 1, 0 1, 0 0, 1 0]  }");
        vis.append("} appearance Appearance { material Material {} texture ImageTexture { repeatS FALSE repeatT FALSE url \"" +
            path + "\"  }    }} ] } ]} ");


        return vis.toString();
    }


    private StringBuffer createMultiColoredVRMLCoords() {


        StringBuffer scheme = new StringBuffer();


        int j = 0;
        while (j < (numRanges - INNER_CUTOFF-1)*12) {

            //scheme.append("0 1 4 5 1");
            scheme.append(j + " " + (j+1) + " " + (j+4) + " " +  (j+5) + " " + (j+1) + " -1 ");
```
189

```
        j++;
        scheme.append(j + " " + (j+1) + " " + (j+4) + " " +  (j+5) + " " + (j+1) + " -1 ");
        j++;
        scheme.append(j + " " + (j+1) + " " + (j+4) + " " +  (j+5) + " " + (j+1) +" -1 ");
        j+=2;
    }

    return scheme;
 }




public String createMultiColoredVRMLRangeRings() {

    int i = 0;
    StringBuffer rings = new StringBuffer();
    StringBuffer colorIndex = new StringBuffer();
    StringBuffer coordIndex = createMultiColoredVRMLCoords();

    StringBuffer rangeMarkers = new StringBuffer();
    while(i < numRadials) { // numRadials
        colorIndex = new StringBuffer();
        int j = INNER_CUTOFF;

        double angle = 2 * i * Math.PI/numRadials;
        double halfAngle = 2 * Math.PI/(numRadials * 2);
        double thirdAngle = 2 * Math.PI/(numRadials * 6);

    rings.append("Transform { rotation 0 1 0 1.57 children [ Shape {  geometry IndexedFaceSet { solid FALSE coordIndex [ " +
            coordIndex.toString() + " ] colorPerVertex FALSE coord Coordinate { point [ ");

        while (j < numRanges) { //numRanges

            double colorR;
            double colorG;
            double colorB;

            int k = 0;

            if (j>INNER_CUTOFF) {
                while (k < 9) {  //9
                    if(grid[i][j][k] >= 30) {
                        colorR = 1; // red
```
190

```
      colorG = 0;

      colorB = 0;

    } else if (grid[i][j][k] >= 25) {

      colorR = 0.949; // orange

      colorG = 0.688;

      colorB = 0.027;

    } else if (grid[i][j][k] >= 20) {

      colorR = 0.949; // dark yellow

      colorG = 0.859;

      colorB = 0.027;

    } else if (grid[i][j][k] >= 15) {

      colorR = 0.935; // yellow

      colorG = 0.910;

      colorB = 0.129;

    } else if (grid[i][j][k] >= 10) {

      colorR = 0.527; // light green

      colorG = 0.867;

      colorB = 0.116;

    } else if (grid[i][j][k] >= 5) {

      colorR = 0.277; // dark green

      colorG = 0.793;

      colorB = 0.191;

    } else if (grid[i][j][k] >= 0) {

      colorR = 0.134; // skyblue

      colorG = 0.776;

      colorB = 0.847;

    } else if (grid[i][j][k] >= -5) {

      colorR = 0.352; // light blue

      colorG = 0.605;

      colorB = 0.618;

    } else if (grid[i][j][k] >= -10) {

      colorR = 0.012; // medium blue

      colorG = 0.129;

      colorB = 0.956;

    } else {

      colorR = 0.449; // dark blue

      colorG = 0.018;

      colorB = 0.953;

    }

    colorIndex.append(colorR + " " + colorG + " " + colorB + " \n");


    k++;

}
```

```
        }

if(j != 0) {
        // add first row of points
        if(j == INNER_CUTOFF) {

                inSpacing = (ranges[1]-ranges[0])*KM_TO_YARDS/2;

                rings.append((Math.cos(angle+halfAngle)*(ranges[j]*KM_TO_YARDS-inSpacing)) + " 0 " +
                        (Math.sin(angle+halfAngle)*(ranges[j]*KM_TO_YARDS-inSpacing)) + ", ");
                rings.append((Math.cos(angle+thirdAngle)*(ranges[j]*KM_TO_YARDS-inSpacing)) + " 0 " +
                        (Math.sin(angle+thirdAngle)*(ranges[j]*KM_TO_YARDS-inSpacing)) + ", ");
                rings.append((Math.cos(angle-thirdAngle)*(ranges[j]*KM_TO_YARDS-inSpacing)) + " 0 " + (Math.sin(angle-
                        thirdAngle)*(ranges[j]*KM_TO_YARDS-inSpacing)) + ", ");
                rings.append((Math.cos(angle-halfAngle)*(ranges[j]*KM_TO_YARDS-inSpacing)) + " 0 " + (Math.sin(angle-
                        halfAngle)*(ranges[j]*KM_TO_YARDS-inSpacing)) + ", ");
        }

        inSpacing = (ranges[j] - ranges[j-1])*KM_TO_YARDS/2;

        if(j == (numRanges-1)) { outSpacing = 250; }
        else { outSpacing = (ranges[j+1] - ranges[j])*KM_TO_YARDS/2; }

        rings.append((Math.cos(angle+halfAngle)*(ranges[j]*KM_TO_YARDS-inSpacing/3)) + " 0 " +
                (Math.sin(angle+halfAngle)*(ranges[j]*KM_TO_YARDS-inSpacing/3)) + ", ");
        rings.append((Math.cos(angle+thirdAngle)*(ranges[j]*KM_TO_YARDS-inSpacing/3)) + " 0 " +
                (Math.sin(angle+thirdAngle)*(ranges[j]*KM_TO_YARDS-inSpacing/3)) + ", ");
        rings.append((Math.cos(angle-thirdAngle)*(ranges[j]*KM_TO_YARDS-inSpacing/3)) + " 0 " + (Math.sin(angle-
                thirdAngle)*(ranges[j]*KM_TO_YARDS-inSpacing/3)) + ", ");
        rings.append((Math.cos(angle-halfAngle)*(ranges[j]*KM_TO_YARDS-inSpacing/3)) + " 0 " + (Math.sin(angle-
                halfAngle)*(ranges[j]*KM_TO_YARDS-inSpacing/3)) + ", ");

        rings.append((Math.cos(angle+halfAngle)*(ranges[j]*KM_TO_YARDS+outSpacing/3)) + " 0 " +
                (Math.sin(angle+halfAngle)*(ranges[j]*KM_TO_YARDS+outSpacing/3)) + ", ");
        rings.append((Math.cos(angle+thirdAngle)*(ranges[j]*KM_TO_YARDS+outSpacing/3)) + " 0 " +
                (Math.sin(angle+thirdAngle)*(ranges[j]*KM_TO_YARDS+outSpacing/3)) + ", ");
        rings.append((Math.cos(angle-thirdAngle)*(ranges[j]*KM_TO_YARDS+outSpacing/3)) + " 0 " + (Math.sin(angle-
                thirdAngle)*(ranges[j]*KM_TO_YARDS+outSpacing/3)) + ", ");
        rings.append((Math.cos(angle-halfAngle)*(ranges[j]*KM_TO_YARDS+outSpacing/3)) + " 0 " + (Math.sin(angle-
                halfAngle)*(ranges[j]*KM_TO_YARDS+outSpacing/3)) + ", ");

        rings.append((Math.cos(angle+halfAngle)*(ranges[j]*KM_TO_YARDS+outSpacing)) + " 0 " +
                (Math.sin(angle+halfAngle)*(ranges[j]*KM_TO_YARDS+outSpacing)) + ", ");
```
192

```
            rings.append((Math.cos(angle+thirdAngle)*(ranges[j]*KM_TO_YARDS+outSpacing)) + " 0 " +
                    (Math.sin(angle+thirdAngle)*(ranges[j]*KM_TO_YARDS+outSpacing)) + ", ");
            rings.append((Math.cos(angle-thirdAngle)*(ranges[j]*KM_TO_YARDS+outSpacing)) + " 0 " + (Math.sin(angle-
                    thirdAngle)*(ranges[j]*KM_TO_YARDS+outSpacing)) + ", ");
            rings.append((Math.cos(angle-halfAngle)*(ranges[j]*KM_TO_YARDS+outSpacing)) + " 0 " + (Math.sin(angle-
                    halfAngle)*(ranges[j]*KM_TO_YARDS+outSpacing)) + ", ");

        }

        j++;
    }

    rings.append((Math.cos(angle-halfAngle)*(ranges[numRanges-1]*KM_TO_YARDS+outSpacing)) + " 0 " + (Math.sin(angle-
        halfAngle)*(ranges[numRanges-1]*KM_TO_YARDS+outSpacing)) + ", " +
        (Math.cos(angle+halfAngle)*(ranges[numRanges-1]*KM_TO_YARDS+outSpacing)) + " 0 " +
        (Math.sin(angle+halfAngle)*(ranges[numRanges-1]*KM_TO_YARDS+outSpacing)));

    rings.append(" ]  }   color Color { color [ " + colorIndex.toString() + "] } } } ]}\n");
    i++;
    }
    return rings.toString();
}

}// end BullsEyePlot
```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX J. SONAR MODELING SCHEMA

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!--W3C Schema generated by XMLSPY v2004 rel. 4 U (http://www.xmlspy.com)-->
<xs:schema elementFormDefault="qualified" xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="BT">
     <xs:complexType>
        <xs:sequence>
           <xs:element ref="BTDepths"/>
           <xs:element ref="BTValues"/>
        </xs:sequence>
        <xs:attribute name="index" type="xs:nonNegativeInteger" use="required"/>
        <xs:attribute name="latitude" type="xs:decimal" use="required"/>
        <xs:attribute name="longitude" type="xs:decimal" use="required"/>
     </xs:complexType>
  </xs:element>
  <xs:element name="BTDepths">
     <xs:complexType>
        <xs:simpleContent>
           <xs:extension base="xs:string">
              <xs:attribute name="units" use="required">
                 <xs:simpleType>
                    <xs:restriction base="xs:string">
                       <xs:enumeration value="km"/>
                       <xs:enumeration value="m"/>
                       <xs:enumeration value="ft"/>
                    </xs:restriction>
                 </xs:simpleType>
              </xs:attribute>
           </xs:extension>
        </xs:simpleContent>
     </xs:complexType>
  </xs:element>
  <xs:element name="BTValues">
     <xs:complexType>
        <xs:simpleContent>
           <xs:extension base="xs:string">
              <xs:attribute name="units" use="required">
                 <xs:simpleType>
```

```xml
                    <xs:restriction base="xs:string">
                        <xs:enumeration value="C"/>
                        <xs:enumeration value="F"/>
                        <xs:enumeration value="K"/>
                    </xs:restriction>
                </xs:simpleType>
            </xs:attribute>
        </xs:extension>
    </xs:simpleContent>
</xs:complexType>
</xs:element>
<xs:element name="Bathymetry">
    <xs:complexType>
        <xs:simpleContent>
            <xs:extension base="xs:string">
                <xs:attribute name="model" type="xs:string" use="required"/>
                <xs:attribute name="units" use="required">
                    <xs:simpleType>
                        <xs:restriction base="xs:string">
                            <xs:enumeration value="km"/>
                            <xs:enumeration value="m"/>
                            <xs:enumeration value="ft"/>
                        </xs:restriction>
                    </xs:simpleType>
                </xs:attribute>
                <xs:attribute name="Reference" use="required">
                    <xs:simpleType>
                        <xs:restriction base="xs:string">
                            <xs:enumeration value="deg"/>
                        </xs:restriction>
                    </xs:simpleType>
                </xs:attribute>
                <xs:attribute name="Resolution" type="xs:decimal" use="required"/>
            </xs:extension>
        </xs:simpleContent>
    </xs:complexType>
</xs:element>
<xs:element name="ComputedData">
    <xs:complexType>
        <xs:sequence>
```

```xml
            <xs:element ref="Depth" maxOccurs="unbounded"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>
<xs:element name="DataStructure">
    <xs:complexType>
        <xs:sequence>
            <xs:element ref="ReceiverLocation"/>
            <xs:element ref="DepthIndex"/>
            <xs:element ref="RadialIndex"/>
            <xs:element ref="RangeIndex"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>
<xs:element name="Depth">
    <xs:complexType>
        <xs:sequence>
            <xs:element ref="Radial" minOccurs="0" maxOccurs="unbounded"/>
        </xs:sequence>
        <xs:attribute name="units">
            <xs:simpleType>
                <xs:restriction base="xs:string">
                    <xs:enumeration value="km"/>
                    <xs:enumeration value="m"/>
                    <xs:enumeration value="ft"/>
                </xs:restriction>
            </xs:simpleType>
        </xs:attribute>
        <xs:attribute name="index" type="xs:nonNegativeInteger"/>
    </xs:complexType>
</xs:element>
<xs:element name="DepthIndex">
    <xs:complexType>
        <xs:sequence>
            <xs:element ref="DepthValue" maxOccurs="unbounded"/>
        </xs:sequence>
        <xs:attribute name="units" use="required">
            <xs:simpleType>
                <xs:restriction base="xs:string">
                    <xs:enumeration value="km"/>
```

197

```
                <xs:enumeration value="m"/>
                <xs:enumeration value="ft"/>
            </xs:restriction>
        </xs:simpleType>
    </xs:attribute>
    <xs:attribute name="count" type="xs:nonNegativeInteger" use="required"/>
  </xs:complexType>
</xs:element>
<xs:element name="DepthValue">
    <xs:complexType>
        <xs:simpleContent>
            <xs:extension base="xs:decimal">
                <xs:attribute name="index" type="xs:nonNegativeInteger" use="required"/>
            </xs:extension>
        </xs:simpleContent>
    </xs:complexType>
</xs:element>
<xs:element name="Description" type="xs:string"/>
<xs:element name="Elevation">
    <xs:complexType>
        <xs:simpleContent>
            <xs:extension base="xs:decimal">
                <xs:attribute name="units" use="required">
                    <xs:simpleType>
                        <xs:restriction base="xs:string">
                            <xs:enumeration value="rad"/>
                            <xs:enumeration value="deg"/>
                        </xs:restriction>
                    </xs:simpleType>
                </xs:attribute>
            </xs:extension>
        </xs:simpleContent>
    </xs:complexType>
</xs:element>
<xs:element name="EnvironmentalData">
    <xs:complexType>
        <xs:sequence>
            <xs:element ref="ReferencePoint"/>
            <xs:element ref="SoundSpeed"/>
            <xs:element ref="SeaTemperature"/>
```
198

```xml
                    <xs:element ref="Bathymetry"/>
                    <xs:element ref="WInd"/>
                    <xs:element ref="SignificantWaveHeight"/>
            </xs:sequence>
            <xs:attribute name="source" type="xs:string" use="required"/>
            <xs:attribute name="timestamp" type="xs:string" use="required"/>
        </xs:complexType>
</xs:element>
<xs:element name="Frequency">
    <xs:complexType>
        <xs:simpleContent>
            <xs:extension base="xs:decimal">
                <xs:attribute name="units" use="required">
                    <xs:simpleType>
                        <xs:restriction base="xs:string">
                            <xs:enumeration value="Hz"/>
                        </xs:restriction>
                    </xs:simpleType>
                </xs:attribute>
            </xs:extension>
        </xs:simpleContent>
    </xs:complexType>
</xs:element>
<xs:element name="HorizontalSweepAngle">
    <xs:complexType>
        <xs:simpleContent>
            <xs:extension base="xs:decimal">
                <xs:attribute name="units" use="required">
                    <xs:simpleType>
                        <xs:restriction base="xs:string">
                            <xs:enumeration value="rad"/>
                            <xs:enumeration value="deg"/>
                        </xs:restriction>
                    </xs:simpleType>
                </xs:attribute>
            </xs:extension>
        </xs:simpleContent>
    </xs:complexType>
</xs:element>
<xs:element name="Latitiude" type="xs:byte"/>
```

```xml
<xs:element name="Latitude">
    <xs:complexType>
        <xs:simpleContent>
            <xs:extension base="LatitudeSimpleType">
                <xs:attribute name="units">
                    <xs:simpleType>
                        <xs:restriction base="xs:string">
                            <xs:enumeration value="UTM"/>
                            <xs:enumeration value="deg"/>
                        </xs:restriction>
                    </xs:simpleType>
                </xs:attribute>
            </xs:extension>
        </xs:simpleContent>
    </xs:complexType>
</xs:element>
<xs:element name="Longitude">
    <xs:complexType>
        <xs:simpleContent>
            <xs:extension base="LongitudeSimpleType">
                <xs:attribute name="units">
                    <xs:simpleType>
                        <xs:restriction base="xs:string">
                            <xs:enumeration value="deg"/>
                            <xs:enumeration value="UTM"/>
                            <xs:enumeration value=""/>
                        </xs:restriction>
                    </xs:simpleType>
                </xs:attribute>
            </xs:extension>
        </xs:simpleContent>
    </xs:complexType>
</xs:element>
<xs:simpleType name="LatitudeSimpleType">
    <xs:restriction base="xs:decimal"/>
</xs:simpleType>
<xs:simpleType name="LongitudeSimpleType">
    <xs:restriction base="xs:decimal"/>
</xs:simpleType>
<xs:element name="ModelDescription">
```

```xml
                <xs:complexType>
                    <xs:sequence>
                        <xs:element ref="ModelType"/>
                        <xs:element ref="URL"/>
                    </xs:sequence>
                </xs:complexType>
        </xs:element>
        <xs:element name="ModelOutput">
                <xs:complexType>
                    <xs:sequence>
                        <xs:element ref="ModelDescription"/>
                        <xs:element ref="DataStructure"/>
                        <xs:element ref="ComputedData"/>
                    </xs:sequence>
                </xs:complexType>
        </xs:element>
        <xs:element name="ModelType" type="xs:string"/>
        <xs:element name="Nomenclature" type="xs:string"/>
        <xs:element name="PressureRatio">
                <xs:complexType>
                    <xs:sequence>
                        <xs:element ref="Ratio" maxOccurs="unbounded"/>
                    </xs:sequence>
                    <xs:attribute name="units" use="required">
                        <xs:simpleType>
                            <xs:restriction base="xs:string">
                                <xs:enumeration value="excess"/>
                                <xs:enumeration value="ratio"/>
                            </xs:restriction>
                        </xs:simpleType>
                    </xs:attribute>
                </xs:complexType>
        </xs:element>
        <xs:element name="RF_Power">
                <xs:complexType>
                    <xs:simpleContent>
                        <xs:extension base="xs:boolean">
                            <xs:attribute name="units" use="required">
                                <xs:simpleType>
                                    <xs:restriction base="xs:string">
```
201

```xml
                    <xs:enumeration value="watts"/>
                </xs:restriction>
            </xs:simpleType>
        </xs:attribute>
    </xs:extension>
</xs:simpleContent>
</xs:complexType>
</xs:element>
<xs:element name="Radial">
    <xs:complexType>
        <xs:sequence>
            <xs:element ref="PressureRatio"/>
        </xs:sequence>
        <xs:attribute name="index" type="xs:nonNegativeInteger" use="required"/>
        <xs:attribute name="rangeCount" type="xs:nonNegativeInteger" use="required"/>
    </xs:complexType>
</xs:element>
<xs:element name="RadialIndex">
    <xs:complexType>
        <xs:sequence>
            <xs:element ref="RadialValue" maxOccurs="unbounded"/>
        </xs:sequence>
        <xs:attribute name="units" use="required">
            <xs:simpleType>
                <xs:restriction base="xs:string">
                    <xs:enumeration value="rad"/>
                    <xs:enumeration value="deg"/>
                </xs:restriction>
            </xs:simpleType>
        </xs:attribute>
        <xs:attribute name="count" type="xs:nonNegativeInteger" use="required"/>
    </xs:complexType>
</xs:element>
<xs:element name="RadialValue">
    <xs:complexType>
        <xs:simpleContent>
            <xs:extension base="RadialValueSimpleType">
                <xs:attribute name="index" type="xs:nonNegativeInteger" use="required"/>
            </xs:extension>
        </xs:simpleContent>
```

202

```
        </xs:complexType>
    </xs:element>
    <xs:simpleType name="RadialValueSimpleType">
        <xs:restriction base="xs:byte">
            <xs:enumeration value="30"/>
            <xs:enumeration value="60"/>
        </xs:restriction>
    </xs:simpleType>
    <xs:element name="RangeIndex">
        <xs:complexType>
            <xs:sequence>
                <xs:element ref="RangeValue" maxOccurs="unbounded"/>
            </xs:sequence>
            <xs:attribute name="units" use="required">
                <xs:simpleType>
                    <xs:restriction base="xs:string">
                        <xs:enumeration value="kyds"/>
                        <xs:enumeration value="yds"/>
                        <xs:enumeration value="nm"/>
                        <xs:enumeration value="km"/>
                    </xs:restriction>
                </xs:simpleType>
            </xs:attribute>
            <xs:attribute name="count" type="xs:nonNegativeInteger" use="required"/>
        </xs:complexType>
    </xs:element>
    <xs:element name="RangeValue">
        <xs:complexType>
            <xs:simpleContent>
                <xs:extension base="RangeValueSimpleType">
                    <xs:attribute name="index" type="xs:nonNegativeInteger" use="required"/>
                </xs:extension>
            </xs:simpleContent>
        </xs:complexType>
    </xs:element>
    <xs:simpleType name="RangeValueSimpleType">
        <xs:restriction base="xs:decimal">
            <xs:enumeration value="0.15242424"/>
            <xs:enumeration value="0.318383838"/>
            <xs:enumeration value="0.456372727"/>
```

203

```xml
          </xs:restriction>
      </xs:simpleType>
  <xs:element name="Ratio">
      <xs:complexType>
          <xs:simpleContent>
              <xs:extension base="RatioSimpleType">
                  <xs:attribute name="rangeIndex" type="xs:nonNegativeInteger" use="required"/>
              </xs:extension>
          </xs:simpleContent>
      </xs:complexType>
  </xs:element>
  <xs:simpleType name="RatioSimpleType">
      <xs:restriction base="xs:decimal">
          <xs:enumeration value="0.23456789"/>
          <xs:enumeration value="1.23456789"/>
          <xs:enumeration value="2.23456789"/>
      </xs:restriction>
  </xs:simpleType>
  <xs:element name="ReceiverLocation">
      <xs:complexType>
          <xs:sequence>
              <xs:element ref="Latitude"/>
              <xs:element ref="Longitude"/>
              <xs:element ref="Depth"/>
          </xs:sequence>
      </xs:complexType>
  </xs:element>
  <xs:element name="Reference" type="xs:anyURI"/>
  <xs:element name="ReferencePoint">
      <xs:complexType>
          <xs:sequence>
              <xs:element ref="Latitiude"/>
              <xs:element ref="Longitude"/>
          </xs:sequence>
          <xs:attribute name="format" use="required">
              <xs:simpleType>
                  <xs:restriction base="xs:string">
                      <xs:enumeration value="deg"/>
                      <xs:enumeration value="UTM"/>
                  </xs:restriction>
```

204

```xml
                </xs:simpleType>
            </xs:attribute>
        </xs:complexType>
    </xs:element>
    <xs:element name="SSP">
        <xs:complexType>
            <xs:sequence>
                <xs:element ref="SSPDepths"/>
                <xs:element ref="SSPValues"/>
            </xs:sequence>
            <xs:attribute name="index" type="xs:nonNegativeInteger" use="required"/>
            <xs:attribute name="latitude" type="xs:decimal" use="required"/>
            <xs:attribute name="longitude" type="xs:decimal" use="required"/>
        </xs:complexType>
    </xs:element>
    <xs:element name="SSPDepths">
        <xs:complexType>
            <xs:simpleContent>
                <xs:extension base="xs:string">
                    <xs:attribute name="units" use="required">
                        <xs:simpleType>
                            <xs:restriction base="xs:string">
                                <xs:enumeration value="m"/>
                                <xs:enumeration value="ft"/>
                            </xs:restriction>
                        </xs:simpleType>
                    </xs:attribute>
                </xs:extension>
            </xs:simpleContent>
        </xs:complexType>
    </xs:element>
    <xs:element name="SSPValues">
        <xs:complexType>
            <xs:simpleContent>
                <xs:extension base="xs:string">
                    <xs:attribute name="units" use="required">
                        <xs:simpleType>
                            <xs:restriction base="xs:string">
                                <xs:enumeration value="kts"/>
                                <xs:enumeration value="m/s"/>
```

```xml
                    </xs:restriction>
                </xs:simpleType>
            </xs:attribute>
        </xs:extension>
    </xs:simpleContent>
</xs:complexType>
</xs:element>
<xs:element name="ScenarioDescription" type="xs:string"/>
<xs:element name="SeaTemperature">
    <xs:complexType>
        <xs:sequence>
            <xs:element ref="BT" maxOccurs="unbounded"/>
        </xs:sequence>
        <xs:attribute name="model" type="xs:string" use="required"/>
    </xs:complexType>
</xs:element>
<xs:element name="SignificantWaveHeight">
    <xs:complexType>
        <xs:simpleContent>
            <xs:extension base="xs:decimal">
                <xs:attribute name="model" type="xs:string" use="required"/>
                <xs:attribute name="units" use="required">
                    <xs:simpleType>
                        <xs:restriction base="xs:string">
                            <xs:enumeration value="ft"/>
                            <xs:enumeration value="m"/>
                        </xs:restriction>
                    </xs:simpleType>
                </xs:attribute>
            </xs:extension>
        </xs:simpleContent>
    </xs:complexType>
</xs:element>
<xs:element name="SonarModeling">
    <xs:complexType>
        <xs:sequence>
            <xs:element ref="ScenarioDescription"/>
            <xs:element ref="ModelOutput"/>
            <xs:element ref="EnvironmentalData"/>
            <xs:element ref="SonarSource" maxOccurs="unbounded"/>
```
206

```xml
            <xs:element ref="Target" maxOccurs="unbounded"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>
<xs:element name="SonarSource">
    <xs:complexType>
        <xs:sequence>
            <xs:element ref="Title"/>
            <xs:element ref="Description"/>
            <xs:element ref="Nomenclature"/>
            <xs:element ref="Reference"/>
            <xs:element ref="SourceProperties"/>
            <xs:element ref="SourceEmployment"/>
        </xs:sequence>
        <xs:attribute name="index" type="xs:nonNegativeInteger" use="required"/>
    </xs:complexType>
</xs:element>
<xs:element name="SoundSpeed">
    <xs:complexType>
        <xs:sequence>
            <xs:element ref="SSP" maxOccurs="unbounded"/>
        </xs:sequence>
        <xs:attribute name="model" type="xs:string" use="required"/>
    </xs:complexType>
</xs:element>
<xs:element name="SourceCourse" type="xs:byte"/>
<xs:element name="SourceDepth">
    <xs:complexType>
        <xs:simpleContent>
            <xs:extension base="xs:decimal">
                <xs:attribute name="units" use="required">
                    <xs:simpleType>
                        <xs:restriction base="xs:string">
                            <xs:enumeration value="ft"/>
                            <xs:enumeration value="m"/>
                        </xs:restriction>
                    </xs:simpleType>
                </xs:attribute>
            </xs:extension>
        </xs:simpleContent>
```

207

```
                </xs:complexType>
        </xs:element>
        <xs:element name="SourceEmployment">
            <xs:complexType>
                <xs:sequence>
                    <xs:element ref="SourceCourse"/>
                    <xs:element ref="SourceSpeed"/>
                    <xs:element ref="SourceLocation"/>
                    <xs:element ref="Elevation"/>
                    <xs:element ref="VerticalSweepAngle"/>
                    <xs:element ref="HorizontalSweepAngle"/>
                </xs:sequence>
            </xs:complexType>
        </xs:element>
        <xs:element name="SourceIntensity">
            <xs:complexType>
                <xs:simpleContent>
                    <xs:extension base="xs:string">
                        <xs:attribute name="units" use="required">
                            <xs:simpleType>
                                <xs:restriction base="xs:string">
                                    <xs:enumeration value="dB"/>
                                </xs:restriction>
                            </xs:simpleType>
                        </xs:attribute>
                    </xs:extension>
                </xs:simpleContent>
            </xs:complexType>
        </xs:element>
        <xs:element name="SourceLocation">
            <xs:complexType>
                <xs:sequence>
                    <xs:element ref="Latitude"/>
                    <xs:element ref="Longitude"/>
                    <xs:element ref="SourceDepth"/>
                </xs:sequence>
                <xs:attribute name="units" use="required">
                    <xs:simpleType>
                        <xs:restriction base="xs:string">
                            <xs:enumeration value="deg"/>
```

208

```xml
                    <xs:enumeration value="UTM"/>
                </xs:restriction>
            </xs:simpleType>
        </xs:attribute>
    </xs:complexType>
</xs:element>
<xs:element name="SourceProperties">
    <xs:complexType>
        <xs:sequence>
            <xs:element ref="SourceIntensity"/>
            <xs:element ref="Frequency"/>
            <xs:element ref="RF_Power"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>
<xs:element name="SourceSpeed">
    <xs:complexType>
        <xs:simpleContent>
            <xs:extension base="xs:decimal">
                <xs:attribute name="units" use="required">
                    <xs:simpleType>
                        <xs:restriction base="xs:string">
                            <xs:enumeration value="kts"/>
                        </xs:restriction>
                    </xs:simpleType>
                </xs:attribute>
            </xs:extension>
        </xs:simpleContent>
    </xs:complexType>
</xs:element>
<xs:element name="Target">
    <xs:complexType>
        <xs:sequence>
            <xs:element ref="TargetDescription"/>
            <xs:element ref="TargetLocation"/>
            <xs:element ref="TargetCourse"/>
            <xs:element ref="TargetSpeed"/>
            <xs:element ref="TargetStrength"/>
        </xs:sequence>
        <xs:attribute name="index" type="xs:nonNegativeInteger" use="required"/>
```

```xml
        </xs:complexType>
    </xs:element>
    <xs:element name="TargetCourse" type="xs:short"/>
    <xs:element name="TargetDepth">
        <xs:complexType>
            <xs:simpleContent>
                <xs:extension base="TargetDepthSimpleType">
                    <xs:attribute name="units" use="required">
                        <xs:simpleType>
                            <xs:restriction base="xs:string">
                                <xs:enumeration value="ft"/>
                                <xs:enumeration value="m"/>
                            </xs:restriction>
                        </xs:simpleType>
                    </xs:attribute>
                </xs:extension>
            </xs:simpleContent>
        </xs:complexType>
    </xs:element>
    <xs:simpleType name="TargetDepthSimpleType">
        <xs:restriction base="xs:short">
            <xs:enumeration value="245"/>
            <xs:enumeration value="60"/>
        </xs:restriction>
    </xs:simpleType>
    <xs:element name="TargetDescription" type="xs:string"/>
    <xs:element name="TargetLatitude" type="xs:decimal"/>
    <xs:element name="TargetLocation">
        <xs:complexType>
            <xs:sequence>
                <xs:element ref="TargetLatitude"/>
                <xs:element ref="TargetLongitude"/>
                <xs:element ref="TargetDepth"/>
            </xs:sequence>
            <xs:attribute name="units" use="required">
                <xs:simpleType>
                    <xs:restriction base="xs:string">
                        <xs:enumeration value="deg"/>
                        <xs:enumeration value="UTM"/>
                    </xs:restriction>
```

```xml
                    </xs:simpleType>
                </xs:attribute>
            </xs:complexType>
        </xs:element>
        <xs:element name="TargetLongitude">
            <xs:simpleType>
                <xs:restriction base="xs:decimal">
                    <xs:enumeration value="-142.2"/>
                    <xs:enumeration value="-145"/>
                </xs:restriction>
            </xs:simpleType>
        </xs:element>
        <xs:element name="TargetSpeed">
            <xs:complexType>
                <xs:simpleContent>
                    <xs:extension base="xs:byte">
                        <xs:attribute name="units" use="required">
                            <xs:simpleType>
                                <xs:restriction base="xs:string">
                                    <xs:enumeration value="kts"/>
                                </xs:restriction>
                            </xs:simpleType>
                        </xs:attribute>
                    </xs:extension>
                </xs:simpleContent>
            </xs:complexType>
        </xs:element>
        <xs:element name="TargetStrength">
            <xs:complexType>
                <xs:simpleContent>
                    <xs:extension base="TargetStrengthSimpleType">
                        <xs:attribute name="units" use="required">
                            <xs:simpleType>
                                <xs:restriction base="xs:string">
                                    <xs:enumeration value="dB"/>
                                </xs:restriction>
                            </xs:simpleType>
                        </xs:attribute>
                    </xs:extension>
                </xs:simpleContent>
```

211

```xml
        </xs:complexType>
    </xs:element>
    <xs:simpleType name="TargetStrengthSimpleType">
        <xs:restriction base="xs:byte">
            <xs:enumeration value="20"/>
            <xs:enumeration value="50"/>
        </xs:restriction>
    </xs:simpleType>
    <xs:element name="Title" type="xs:string"/>
    <xs:element name="URL" type="xs:anyURI"/>
    <xs:element name="U_Component" type="xs:decimal"/>
    <xs:element name="V_Component" type="xs:decimal"/>
    <xs:element name="VerticalSweepAngle">
        <xs:complexType>
            <xs:simpleContent>
                <xs:extension base="xs:decimal">
                    <xs:attribute name="units" use="required">
                        <xs:simpleType>
                            <xs:restriction base="xs:string">
                                <xs:enumeration value="rad"/>
                                <xs:enumeration value="deg"/>
                            </xs:restriction>
                        </xs:simpleType>
                    </xs:attribute>
                </xs:extension>
            </xs:simpleContent>
        </xs:complexType>
    </xs:element>
    <xs:element name="WInd">
        <xs:complexType>
            <xs:sequence>
                <xs:element ref="U_Component"/>
                <xs:element ref="V_Component"/>
            </xs:sequence>
            <xs:attribute name="model" type="xs:string" use="required"/>
            <xs:attribute name="units" use="required">
                <xs:simpleType>
                    <xs:restriction base="xs:string">
                        <xs:enumeration value="kts"/>
                        <xs:enumeration value="m/s"/>
```

```
            </xs:restriction>
         </xs:simpleType>
      </xs:attribute>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX K. EXAMPLE SONAR MODELING XML FILE

```xml
<?xml version="1.0" encoding="UTF-8"?> <?xml version="1.0" encoding="UTF-8"?>
    <SonarModeling xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                    xsi:noNamespaceSchemaLocation="C:\metcast\Composite\SonarModelingFinal.xsd">
        <ScenarioDescription>Here we would describe the scenario</ScenarioDescription>
            <ModelOutput>
                <ModelDescription>
                    <ModelType>CASS/GRAB</ModelType>
                    <URL>http://www.weaver.navy.mil</URL>
                </ModelDescription>
                <DataStructure>
                    <ReceiverLocation>
                        <Latitude units="UTM">14.5</Latitude>
                        <Longitude units="UTM">-144.5</Longitude>
                        <Depth units="ft"/>
                    </ReceiverLocation>
                    <DepthIndex units="km" count="2">
                        <DepthValue index="0">0.01456382828</DepthValue>
                        <DepthValue index="1">0.01456382828</DepthValue>
                    </DepthIndex>
                    <RadialIndex units="deg" count="2">
                        <RadialValue index="0">30</RadialValue>
                        <RadialValue index="1">60</RadialValue>
                    </RadialIndex>
                    <RangeIndex units="km" count="3">
                        <RangeValue index="0">0.15242424</RangeValue>
                        <RangeValue index="1">0.318383838</RangeValue>
                        <RangeValue index="2">0.456372727</RangeValue>
                    </RangeIndex>
                </DataStructure>
                <ComputedData>
                    <Depth index="0">
                        <Radial index="0" rangeCount="3">
                            <PressureRatio units="ratio">
                                <Ratio rangeIndex="0">1.23456789</Ratio>
                                <Ratio rangeIndex="1">0.23456789</Ratio>
                                <Ratio rangeIndex="2">2.23456789</Ratio>
                            </PressureRatio>
```

215

```xml
            </Radial>
            <Radial index="1" rangeCount="3">
                <PressureRatio units="ratio">
                    <Ratio rangeIndex="0">1.23456789</Ratio>
                    <Ratio rangeIndex="1">0.23456789</Ratio>
                    <Ratio rangeIndex="2">2.23456789</Ratio>
                </PressureRatio>
            </Radial>
            <Radial index="2" rangeCount="3">
                <PressureRatio units="ratio">
                    <Ratio rangeIndex="0">1.23456789</Ratio>
                    <Ratio rangeIndex="1">0.23456789</Ratio>
                    <Ratio rangeIndex="2">2.23456789</Ratio>
                </PressureRatio>
            </Radial>
        </Depth>
        <Depth index="1">
            <Radial index="0" rangeCount="3">
                <PressureRatio units="ratio">
                    <Ratio rangeIndex="0">1.23456789</Ratio>
                    <Ratio rangeIndex="1">0.23456789</Ratio>
                    <Ratio rangeIndex="2">2.23456789</Ratio>
                </PressureRatio>
            </Radial>
            <Radial index="1" rangeCount="3">
                <PressureRatio units="ratio">
                    <Ratio rangeIndex="0">1.23456789</Ratio>
                    <Ratio rangeIndex="1">0.23456789</Ratio>
                    <Ratio rangeIndex="2">2.23456789</Ratio>
                </PressureRatio>
            </Radial>
            <Radial index="2" rangeCount="3">
                <PressureRatio units="ratio">
                    <Ratio rangeIndex="0">1.23456789</Ratio>
                    <Ratio rangeIndex="1">0.23456789</Ratio>
                    <Ratio rangeIndex="2">2.23456789</Ratio>
                </PressureRatio>
            </Radial>
        </Depth>
    </ComputedData>
```

```xml
</ModelOutput>
<EnvironmentalData source="Metcast" timestamp="121235SEP2004">
    <ReferencePoint format="UTM">
        <Latitiude>15</Latitiude>
        <Longitude>-145</Longitude>
    </ReferencePoint>
    <SoundSpeed model="OTIS_GLOBAL">
        <SSP index="1" latitude="14.33" longitude="-144.83">
            <SSPDepths units="ft">0 50 100 200</SSPDepths>
            <SSPValues units="m/s">1500.8 1501.3 1502 .11501.2</SSPValues>
        </SSP>
        <SSP index="2" latitude="14.66" longitude="-144.83">
            <SSPDepths units="ft">0 50 100 200</SSPDepths>
            <SSPValues units="m/s">1500.8 1501.3 1502 .11501.2</SSPValues>
        </SSP>
    </SoundSpeed>
    <SeaTemperature model="OTIS_GLOBAL">
        <BT index="1" latitude="14.33" longitude="-144.83">
            <BTDepths units="ft">0 50 100 200</BTDepths>
            <BTValues units="K">1500.8 1501.3 1502 .11501.2</BTValues>
        </BT>
        <BT index="2" latitude="14.66" longitude="-144.83">
            <BTDepths units="ft">0 50 100 200</BTDepths>
            <BTValues units="K">1500.8 1501.3 1502 .11501.2</BTValues>
        </BT>
    </SeaTemperature>
    <Bathymetry model="ETOPO2" units="m" Reference="deg"
            Resolution="0.03333333333333333">0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0</Bathymetry>
    <WInd model="NOGAPS" units="m/s">
        <U_Component>1.5</U_Component>
        <V_Component>5.6</V_Component>
    </WInd>
    <SignificantWaveHeight model="WW3_GLOBAL" units="m">3.4</SignificantWaveHeight>
</EnvironmentalData>
<SonarSource index="1">
    <Title>DICASS Sonobuoy</Title>
    <Description>The DICASS sonobuoy is an active sonobuoy</Description>
    <Nomenclature>AN/SSQ-62D</Nomenclature>
    <Reference>http://www.sparton.com/service/markets/sonobuoy.htm</Reference>
    <SourceProperties>
```

```xml
        <SourceIntensity units="dB">190.0 </SourceIntensity>
        <Frequency units="Hz">9.5</Frequency>
        <RF_Power units="watts">1</RF_Power>
    </SourceProperties>
    <SourceEmployment>
        <SourceCourse>015</SourceCourse>
        <SourceSpeed units="kts">0.2</SourceSpeed>
        <SourceLocation units="UTM">
            <Latitude>15.0</Latitude>
            <Longitude>-145.0</Longitude>
            <SourceDepth units="ft">200.0</SourceDepth>
        </SourceLocation>
        <Elevation units="deg">45.0</Elevation>
        <VerticalSweepAngle units="deg">30.0</VerticalSweepAngle>
        <HorizontalSweepAngle units="deg">360.0</HorizontalSweepAngle>
    </SourceEmployment>
</SonarSource>
<SonarSource index="2">
    <Title>DICASS Sonobuoy</Title>
    <Description>The DICASS sonobuoy is an active sonobuoy</Description>
    <Nomenclature>AN/SSQ-62D</Nomenclature>
    <Reference>http://www.sparton.com/service/markets/sonobuoy.htm</Reference>
    <SourceProperties>
        <SourceIntensity units="dB">190.0 </SourceIntensity>
        <Frequency units="Hz">9.5</Frequency>
        <RF_Power units="watts">1</RF_Power>
    </SourceProperties>
    <SourceEmployment>
        <SourceCourse>015</SourceCourse>
        <SourceSpeed units="kts">0.2</SourceSpeed>
        <SourceLocation units="UTM">
            <Latitude>15.0</Latitude>
            <Longitude>-145.0</Longitude>
            <SourceDepth units="ft">200.0</SourceDepth>
        </SourceLocation>
        <Elevation units="deg">45.0</Elevation>
        <VerticalSweepAngle units="deg">30.0</VerticalSweepAngle>
        <HorizontalSweepAngle units="deg">360.0</HorizontalSweepAngle>
    </SourceEmployment>
</SonarSource>
```

218

```xml
    <Target index="1">
        <TargetDescription>Los Angeles class submarine</TargetDescription>
        <TargetLocation units="UTM">
            <TargetLatitude>13.34</TargetLatitude>
            <TargetLongitude>-145</TargetLongitude>
            <TargetDepth units="ft">60</TargetDepth>
        </TargetLocation>
        <TargetCourse>315</TargetCourse>
        <TargetSpeed units="kts">12</TargetSpeed>
        <TargetStrength units="dB">50</TargetStrength>
    </Target>
    <Target index="2">
        <TargetDescription>Diesel 209</TargetDescription>
        <TargetLocation units="UTM">
            <TargetLatitude>13.34</TargetLatitude>
            <TargetLongitude>-142.2</TargetLongitude>
            <TargetDepth units="ft">245</TargetDepth>
        </TargetLocation>
        <TargetCourse>315</TargetCourse>
        <TargetSpeed units="kts">12</TargetSpeed>
        <TargetStrength units="dB">20</TargetStrength>
    </Target>
</SonarModeling>
```

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF REFERENCES

Beatty, Bill. "Interactive Mulitsensor Analysis Training." *Wavelengths,* January/February 1999. Http://www.d-a-s.com/IMATexcerpt.html. Accessed 1 August 2004.

Brutzman, Don. "A Virtual World for an Autonomous Underwater Vehicle." Ph.D. diss., Doctoral Dissertation, Naval Postgraduate School, 1994.

Card, Stuart, Jock Mackinlay, and Ben Shneiderman. *Readings in Information Visualization: Using Vision to Think.* San Diego, CA: Academic Press, 1999.

Cerami, Ethan. *Web Services Essentials*. Sebastopol, CA: O'Reilly & Associates, 2002.

Chopra, Vivek, Ben Galbraith, and Sing Li, eds. *Professional Apache Tomcat.* Indianapolis, IN: Wiley Publishing, 2003.

Chu, Peter, Carlos Cintron, and Steven Haeger, eds. "Acoustic Mine Detection Using the Navy's CASS/GRAB Model." *Journal of Counter-Ordnance Technology (*2002). Http://www.oc.nps.navy.mil/~chu/web_paper /jcot/cassgrab.pdf. Accessed 1 August 2004.

Chu, Peter, Nick Vares, and Ruth Keenan. "Uncertainty in Acoustic Mine Detection Due To Environmental Variabillity." *Journal of Counter-Ordnance Technology* (May 2004). Http://www.oc.nps.navy.mil/~chu/web_paper/proceedings/jcot/acoustic.pdf. Accessed 1 Aug 2004.

Clark, Vern. "Sea Power 21: Projecting Decisive Joint Capabilities." *Proceedings* October 2002. Http://www.chinfo.navy.mil/navpalib/cno./proceedings/html. Accessed 1 Aug 2004.

Etter, Paul. "Underwater Acoustic Modeling and Simulation." London: Spon Press, 2003.

Holliday, Timothy. "Real-Time 3D Sonar Modeling and Visualization." Master's Thesis, Naval Postgraduate School, 1998.

http://java.sun.com/. Accessed 3 September 2004.

http://sourceforge.net/. Accessed 3 September 2004.

http://web.nps.navy.mil/~brutzman/Savage. Accessed 13 September 2004.

http://www.w3.org/XML/1999/XML-in-10-points. Accessed 21 September 2004.

http://www.apache.org/. Accessed 3 September 2004.

http://www.dtic.mil/ndia/2001interop/thompson.pdf. Accessed 12 September 2004.

https://www.fnmoc.navy.mil/PUBLIC. Accessed 12 September 2004.

http://www.innovation.ch/java/HTTPClient/. Accessed 3 September 2004.

http://www.jfree.org/jfreechart/. Accessed 3 September 2004.

http://www.onr.navy.mil. Accessed 21 September 2004.

http://www.onr.navy.mil/sci_tech/personnel/342/training/majapps/imat. Accessed 12
        September 2004.

http://www.parallelgraphics.com/products/cortona. Accessed 3 September 2004.

http://www.unidata.ucar.edu/packages/netcdf/index.html. Accessed 3 September 2004.

http://www.web3d.org/x3d/. Accessed 3 September 2004.

http://www.xj3d.org/. Accessed 3 September 2004.

Hunter, David, Kurt Cagle, and Chris Dix, eds. *Beginning XML, Second Edition*.
        Indianapolis, IN: Wiley Publishing, 2003.

Li, Jianwei, Wei-keng Liao and Alok Choudhary, eds. "Parallel netCDF: A High-
        Performance Scientific I/O Interface." *International Conference for High
        Performance Computing and Communications,* 2003. Http://www.sc-
        conference.org/sc2003/ paperpdfs/pap258.pdf. Accessed 1 Aug 2004.

Mayo, Richard and John Nathman. "ForceNet: Turning Information into Power."
        Proceedings, February 2003. Http://www.usni.org/Proceedings/Articles03/
        PROmayo02.htm. Accessed 1 August
        2004.

Norman, D.A. *Things That Make Us Smart.* Reading, MA: Addison-Wesley 1993.

Ruhston, Richard, Michael McCrave, and Mark Klett, eds. *Open Architecture,
        The Critical Network Centric Warfare Enabler, Second Edition* (2004).
        Http://kcg-inc.net/OPNAV_766/docs/NCWed2.pdf. Accessed 1 Aug, 2004.

Sauter, David and Mario Torres. *Employing Net Centric Technology for a Mobile
        Weather Intelligence Capability.* Army Research Laboratory, NM (2004).
        Http://www.dodccrp.org/events/2004/CCRTS_San_Diego/CD/
        papers/040.pdf. Accessed 1 August 2004.

Swallow, Tina, and Danelle Barrett. "Transforming the Navy with Web Technology."

Chips, Spring 2003. Http://www.chips.navy.mil/archives/03_spring/ PDF/chipspring03.pdf. Accessed 1 Aug 2004.

Urick, Robert. *Principles of Underwater Sound, Second Edition*. New York, NY: McGraw Hill, 1975.

Ziomek, Lawrence. "The RRA Algorithm: Recursive Ray Acoustics for Three-Dimensional Speeds of Sound." *IEEE Journal of Oceanic Engineering*, vol. 18 no. 1, (January 1993).

Ziomek, Lawrence. *Fundamentals of Acoustic Field Theory and Space-Time Signal Processing, First Edition*. Boca Raton, FL: CRC Press, 1995

Zyda, Mike, Don Brutzman, and Mark Pullen, eds. "Extensible Modeling and Simulation Framework (XMSF): Challenges for Web-Based Modeling and Simulation." *Strategic Opportunities Symposium*. (October 2002).

THIS PAGE INTENTIONALLY LEFT BLANK

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
    Ft. Belvoir, Virginia

2. Dudley Knox Library
    Naval Postgraduate School
    Monterey, California

3. RDML Patrick Dunne USN
    Naval Postgraduate School
    Monterey, CA

4. Dr. Donald P. Brutzman
    Naval Postgraduate School
    Monterey, CA

5. LT Scott A. Rosetti USN
    U.S. Navy
    Chesterton, IN

6. Erik Chaum
    NUWC
    Newport, RI

7. David Bellino
    NPRI
    Newport, RI

8. Dick Nadolink
    NUWC
    Newport, RI

9. VADM Roger Bacon (Ret.)
    Naval Postgraduate School
    Monterey, CA

10. K.C. Stangl
    NAVAIR
    Patuxent River, MD

11. Margaret Bailey
    Sonalysts
    Waterford, CT

12. Dr. Peter Chu
   Naval Postgraduate School
   Monterey, CA

13. RDML Elizabeth Hight USN
   OPNAV 61 – C4
   Washington, DC

14. Doug Backes
   COMPACFLT Science Advisor
   Pearl Harbor, HI

15. LT Andrew Hurvitz USN
   FNMOC
   Monterey , CA

16. ENS Darin Keeter USN
   FNMOC
   Monterey , CA

17. CAPT David Titley USN
   FNMOC
   Monterey, CA

18. CAPT Scot Miller USN
   NCTSI
   San Diego, CA

19. RADM Paul Sullivan USN
   COMSUBPAC
   Pearl Harbor, HI

20. CAPT Arn Lotring Jr. USN
   Submarine Learning Center
   Groton, CT

21. Dr. Alan Washburn
   Naval Postgraduate School
   Monterey, CA

22. Hans Widmer
   COMSUBPAC Science Advisor
   Pearl Harbor, HI

23.   CAPT Bill Burke USN
             Office of VCNO
             Washington, DC