NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

XML TACTICAL CHAT (XTC):
EXTENSIBLE MESSAGING AND PRESENCE PROTOCOL FOR COMMAND AND CONTROL APPLICATIONS

by

Adrian D. Armold

September 2006

Thesis Advisor:	Don Brutzman
Co-Advisor:	Don McGregor
Second Reader:	Terry Norbraten

Approved for public release; distribution is unlimited

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 2006	**3. REPORT TYPE AND DATES COVERED** Master's Thesis	
4. TITLE AND SUBTITLE: XML Tactical Chat (XTC): Extensible Messaging and Presence Protocol for Command and Control Applications			**5. FUNDING NUMBERS**
6. AUTHOR Adrian D. Armold			
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000			**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING /MONITORING AGENCY NAME(S) AND ADDRESS(ES) COSMOS ACTD, OSD –ATL 3400 Defense Pentagon, Washington, DC 20301- 3400 Navy Modeling and Simulation Office 1333 Isaac Hull Ave., Stop 5012, Washington Navy Yard, D.C. 20376-5012			**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

Current chat and instant messaging (IM) solutions within the DoD have created problems with information security and interoperability. Though Extensible Message and Presence Protocol (XMPP) is the only mandated chat and IM protocol in the DoD, the majority of the military still operates alternate nonstandard solutions that prevent interoperability and lack appropriate security assurances.

XMPP is a streaming XML protocol used for multi-user text chat and Instant Messaging (IM). XMPP supports a large set of administrative and user features, valuable to military chat and IM users. As an open standard, XMPP is also extensible to allow for development of military-specific chat and IM requirements. XMPP protocol also provides significant extensibility to allow for greater command and control and other operational capabilities.

This work demonstrates the use of XMPP to route XML-expressed Distributed Interactive Simulation (DIS-XML) data to conduct distributed modeling and simulation. This work also demonstrates the use of XMPP as a generalized XML message-routing framework in conjunction with XML-expressed military data models, such as the Joint Consultation Command and Control Information Exchange Data Model. Also presented in this thesis is an XML document based chat data logger, designed to support persistent operations using distributed chat architecture.

Experiments conducted with Navy Exercise Trident Warrior 2006 demonstrate the value of such a framework, as well as the value of XML document-based chat data logging. Results indicate that implementation and extension of XMPP has significant value for enhancing command and control. These features, along with the benefits of the adoption of open standard solutions, make XMPP an essential technology for adoption in today's operational command and control suites.

14. SUBJECT TERMS Instant Messaging, Chat, XMPP, XML, DIS-XML, DIS, JC3IEDM, Chat Logging, Command and Control, XTC, X3D, XMSF	**15. NUMBER OF PAGES** 192
	16. PRICE CODE

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

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

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release; distribution is unlimited

XML TACTICAL CHAT (XTC): EXTENSIBLE MESSAGING AND PRESENCE PROTOCOL FOR COMMAND AND CONTROL APPLICATIONS

Adrian D. Armold
Captain, United States Marine Corps
B.A., University of Colorado-Boulder, 1996

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
September 2006

Author: Adrian D. Armold

Approved by: Don Brutzman
Thesis Advisor

Donald R. McGregor
Co-Advisor

Terry Norbraten
Second Reader

Peter J. Denning
Chair, Department of Computer Science

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

Current chat and instant messaging (IM) solutions within the DoD have created problems with information security and interoperability. Though Extensible Message and Presence Protocol (XMPP) is the only mandated chat and IM protocol in the DoD, the majority of the military still operates alternate nonstandard solutions that prevent interoperability and lack appropriate security assurances.

XMPP is a streaming XML protocol used for multi-user text chat and Instant Messaging (IM). XMPP supports a large set of administrative and user features, valuable to military chat and IM users. As an open standard, XMPP is also extensible to allow for development of military-specific chat and IM requirements. XMPP protocol also provides significant extensibility to allow for greater command and control and other operational capabilities.

This work demonstrates the use of XMPP to route XML-expressed Distributed Interactive Simulation (DIS-XML) data to conduct distributed modeling and simulation. This work also demonstrates the use of XMPP as a generalized XML message-routing framework in conjunction with XML-expressed military data models, such as the Joint Consultation Command and Control Information Exchange Data Model. Also presented in this thesis is an XML document based chat data logger, designed to support persistent operations using distributed chat architecture.

Experiments conducted with Navy Exercise Trident Warrior 2006 demonstrate the value of such a framework, as well as the value of XML document-based chat data logging. Results indicate that implementation and extension of XMPP has significant value for enhancing command and control. These features, along with the benefits of the adoption of open standard solutions, make XMPP an essential technology for adoption in today's operational command and control suites.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

I would like to thank Don Brutzman for his guidance and patience in the conduct of this work. His vision, enthusiasm, and dedication to enhancing our military are tremendous, and his contributions to the DoD and the warfighter are invaluable.

I owe many thanks to Don McGregor for his support in this endeavor as well. The research for this thesis could not have been conducted without his technical expertise and continuing administrative and system hardware support.

Additional thanks go to Terry Norbraten for tireless assistance in the preparations for Trident Warrior 2006, and for conducting valuable research in my place during my wedding. His selfless dedication to his and his peer's work is inspirational.

I would like to acknowledge the efforts of the Captain Burns, Commander Krissa Baylor, and Lieutenant Commander Randy Conley of the ONR/NRL Detachment 113 for their contribution to the JTC experiment with Navy Exercise Trident Warrior 2006.

I would like to acknowledge the COSMOS ACTD and the Navy Modeling and Simulation Office for research funding in support of this thesis.

Final thanks go to my wife Tracey, whose patience, caring, and support helped me in more ways than can be expressed.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

A. PROBLEM STATEMENT

Chat technology has emerged as a significant ad hoc solution to the problem of real-time information distribution among today's deployed military. The real-time information requirements of military operations today are quite large and chat has been widely adopted as a technology of choice for conducting command and control activities in support of military missions. Chat is ubiquitously used to develop plans, coordinate operations, conduct autonomous vehicle missions, and support collaborative decision making.

Rapid and uncoordinated adoption of incompatible chat protocols has led to the disjoint application of chat technology across the services, and there are significant problems with joint interoperability with respect to chat. The individual services have separate preferences with regard to chat tools and clients. There exists no overarching architecture for chat across the Department of Defense (DoD). There are limited official chat requirement documents at either the service or the joint level, and the emergence of any official service sponsorship of a chat system is only recent (Eovito, 2006).

The terms Chat, Group Chat, Multi-User Chat (MUC) and Instant Messaging have varying degrees of overlapping meaning. Instant Messaging is thought of as one-to-one text communication while Group Chat is many-to-many text communication. "Chat" is sometimes used to refer to either one of the two communication types. Both communication methods are typically provided by most Instant Message service providers, but the majority of use in the military context is of the Multi-User Chat type.

The preeminent chat system solution today is the Extensible Messaging and Presence Protocol (XMPP). It is Internet Engineering Task Force (IETF) instant messaging and presence standard. XMPP is an open-standard solution to the myriad needs and implementations of text-based chat in the military. As stated in the IETF Request for Comment (RFC) 3920 , XMPP is "…a protocol for streaming Extensible Markup Language (XML) elements in order to exchange structured information in close

to real time between any two network endpoints." (IETF, Network Working Group, 2004), this protocol provides enormous extensibility in the application of the XMPP framework beyond simple text-based chat.

This thesis identifies the current uses and requirements of chat in the military, and describes the XMPP protocol systems as a solution for future chat systems.

This research also exhibits the value of the flexibility and extensibility of the XMPP framework; presenting exemplar applications in the areas of structured message passing, message archiving and searching. This is done to demonstrate the greater potential that an XMPP-based chat solution has over other chat solutions. These exemplars also expose such extensions as real-value additions that go beyond traditional unstructured text-based chat.

In sum, the following questions are addressed. Is an XMPP based Chat/IM solution capable of meeting the requirements of today's military? What advantages, if any, does adopting the open XMPP standard based solution hold over alternate solutions? How can XML streams and XMPP extensions be applied to increase the capabilities value of Chat/IM communications for Command and Control (C2)?

B. MOTIVATION

The motivation for this thesis lies in identifying XMPP as the well-engineered synthesis of two distinct technologies, XML and Chat/IM. Chat/IM is already established as a critical communication technology. Eovito claims that Chat has "…migrated from a stopgap measure, the proverbial finger in the dike, and become one of the main real-time C2 systems used by Commanders and operators to execute all phases of their doctrinal missions" (Eovito, 2006). However, the ad hoc adoption of incompatible chat protocols across the services, has created a problem space. The lack of systematic requirements elicitation and documentation has not allowed for informed selection of chat tools. Further, the disparate solutions currently used lack the interoperability needed in such a critical C2 communications method.

XML, though relatively new, has matured rapidly and is well established as a format for structuring and handling data. XML is platform independent, well-supported,

and does not require a license (Bos, 2000). It is an underlying technology for the World Wide Web, and it and web services "enable a paradigm shift in integration and interoperability" in the context of military C2 (Molitoris, 2003).

XMPP provides a technology that can both meet the emerging requirements for Chat and IM and leverage the advantages of XML. The motivation for this research lies in exploring the potential that XMPP technology holds for improving and expanding the C2 capabilities enabled by chat and IM.

C. OBJECTIVES

This thesis seeks to assess the suitability of XMPP-based Chat to military systems, and to explore and demonstrate the potential enhancements to C2 that streaming XML affords the DoD.

An analysis of the XMPP specifications against the military requirements for chat and IM is presented.

A presentation of existing XMPP implementations is made. Both server and client applications are exposed and assessed for their suitability for military use.

This thesis presents exemplars of XML-based message passing over XMPP as a demonstration of the extensible power that the protocol holds. This message passing capability is examined in the context of military modeling and simulation (M&S) and military C2.

Finally, this thesis presents an exemplar system of recording, archiving, and searching on XMPP based Chat conversations. Again, the extensibility of XML and XMPP is highlighted.

D. THESIS ORGANIZATION

This thesis is comprised of eight chapters:

- Chapter I – Introduction. This chapter provides an overview of the topic and the motivation and objectives of the research.
- Chapter II – Background and Related Work. This chapter provides background discussion on XML and Chat/IM., and reviews related work in the fields of Distributed Interactive Simulation protocol (DIS), Joint Consultation Command and Control Information Exchange Model (JC3IEDM), XML Binary Serialization, and search/data mining.

- Chapter III – Military Chat. This chapter provides an overview of the status of Chat/IM in the DoD with respect to requirements.
- Chapter IV – Extensible Messaging and Presence Protocol. This chapter provides an overview of the XMPP technology for Chat/IM.
- Chapter V – XMPP Deployment and Implementation. This chapter provides discussion of the XMPP client applications available today, and discusses technical and administrative issues related to XMPP server deployment.
- Chapter VI – XML Tactical Chat. This chapter discusses and demonstrates the application of XML-based messaging to XMPP Chat/IM and the suitability of XMPP to military networks. Additionally, this chapter discusses and demonstrates Chat archiving and retrieval.
- Chapter VII – Experimental Results and Applications. This chapter presents the results of XTC related research and applications.
- Chapter VIII – Conclusions and Recommendations. This chapter summarizes conclusions and discusses future work.
- Appendices – The appendices provide a list of references, user guides for applications, a Navy message regard the results of a chat experiment, and documentation for the Trident Warrior 2006 exercise.

II. BACKGROUND AND RELATED WORK

A. INTRODUCTION

XMPP technology and tools comprise a convergence of two previously disparate technologies, Instant Messaging/Presence and XML. This chapter presents an overview and background of these two technologies. Overviews on the related work fields of Distributed Interactive Simulation (DIS), Joint Consultation Command and Control Information Exchange Model (JC3IEDM), XML binary serialization, and information retrieval, or search, is presented as well.

B. BACKGROUND

1. Chat and Instant Messaging (IM)

Chat and Instant Messaging (IM) are methods of communication which pass text messages from one computer user to another in near-real time. IM describes such communication between exactly two people, while chat describes this communication between multiple users. Typically, these messages are displayed in a graphic user interface (GUI) which identifies the sender of a message, the time that the message was sent, and the text message itself. A scrolling pane of messages is maintained for an Chat/IM session. Figure 1 displays a typical Chat/IM user interface.

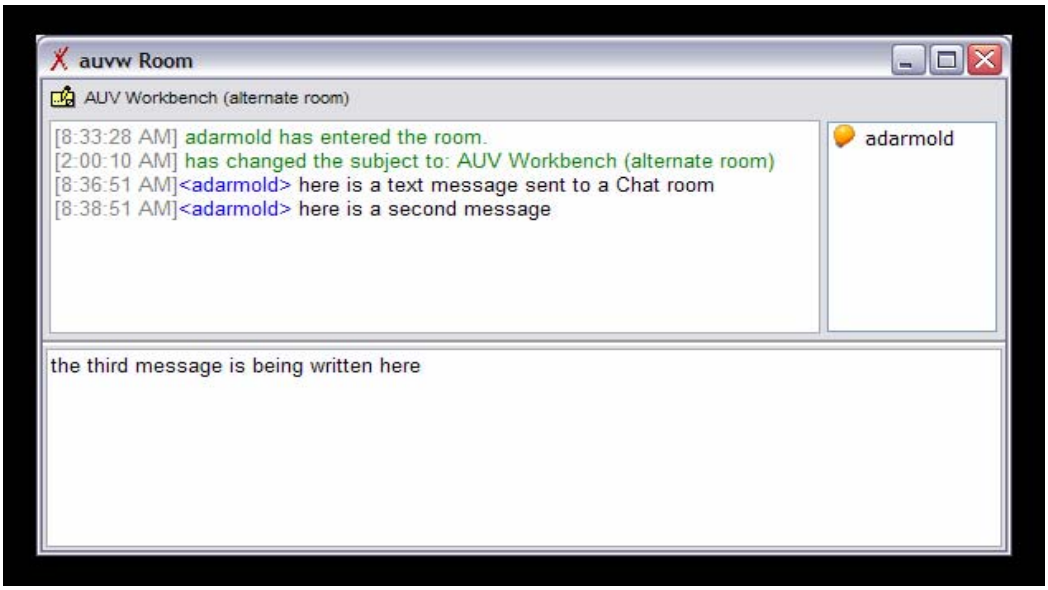

Figure 1. Chat interfaces typically display a list of users, a scrolling display of sent messages and a window for composing text messages.

Chat/Instant Messaging has one of its origins in the `talk` program feature of the Unix operating system in the 1970s. Originally, Unix `talk` program supported text communications between two users of a single multi-user computer, though later versions provided communication between multiple users and users on different machines (Wikipedia.org, 2006c). The advent of Chat/IM as an OS independent application was first seen with Bitnet Relay Chat, or Relay, in 1985 by Jeff Kell (Kell, 1987). Bitnet was a computer network used by American universities from 1981 until 1991. Bitnet dissolved with the widespread adoption of the Internet in the early 1990's. The first Chat/IM system to be implemented on the Internet was the Internet Relay Chat (IRC).

a. Internet Relay Chat (IRC)

IRC was released by Jarkko Oikarinen in August, 1988 (Oikarinen, 2005). In May 1993, IRC was standardized by the informational IETF RFC 1459 (Oikarinen and Reed, 1993). The standard has been updated and today IRC is standardized by informational IETF RFCs 2810: IRC Architecture (Kalt, 2000d), 2811: IRC Channel Management (Kalt, 2000a), 2812: IRC Client Protocol (Kalt, 2000b) , and 2813: IRC Server Protocol (Kalt, 2000c). IRC is still widely used today, and is one of the most commonly used Chat/IM systems in the DoD (Eovito, 2006). IRC uses client/server architecture. Groups of servers comprise an IRC network, and users, through a client program, connect to a server on the network. Once connected, users may chat with other connected users privately or enter a "channel" which is a multi-user chat room dedicated to a particular topic or group. Though there are many very large IRC networks, not all IRC servers are connected to each other. As a result of various disputes over technical and administrative decisions about IRC, the original IRC network split into two large disconnected IRC networks in 1996. Today, hundreds of smaller IRC networks exist worldwide. There are both proprietary and open-source server and client side applications in support of Windows, Unix, and other operating systems. IRC suffers from two major architectural problems. First, all servers in an IRC network must know about all other servers, and second, IRC does not support user authentication or encryption.

b. America OnLine (AOL) Instant Messenger (AIM)

AIM is one of the older Chat/IM services, originating as a messaging service for AOL users. In 1996, ICQ IM became a free alternative to AIM, resulting in

AOL's acquisition of ICQ and its company of origin, Mirabilis. In 2002, AOL obtained patents on ICQ, but has done little to enforce them as many companies, such as Yahoo and MSN and have produced similar systems (Wikipedia.org, 2006a). AIM operates using the Open System CommunicAtion in Realtime (OSCAR) protocol. This is a proprietary protocol whose documentation and sample code has never been released by AOL. OSCAR is also client/server architecture, and users must connect to an AOL server. There are multiple implementations of the AIM protocol which have been reverse-engineered from the AOL supplied products. In June of 2006, AOL converted their client application to open source and released a development kit for programmers. AIM currently has approximately 155 million account holders (Sanders, 2006).

c. Windows Messenger / Yahoo Messenger

Microsoft (Windows Live Messenger) and Yahoo (Yahoo Messenger with Voice) each offer their own IM service, but in October of 2005, announced that the companies will be modifying their technologies to allow for interoperability (Sanders, 2006). Windows Live Messenger is built on the Mobile Status Notification Protocol (MSNP) and Yahoo Messenger with Voice is built on Yahoo Messenger Protocol and YMSG. A beta version of the interoperable software was released on 13 July, 2006. Combined, the two systems form the second largest IM network behind AIM with 120 million users (Sanders, 2006).

d. Session Initiation Protocol for Instant Message and Presence Leveraging Extensions (SIMPLE)

SIMPLE is an open standard protocol for chat and IM. It is based on the application of presence and IM mechanisms to the Session Initiation Protocol. SIMPLE is still undergoing the IETF standardization process. Microsoft and other IM vendors have elected to support SIMPLE as the IM and presence standard, but Microsoft's Windows Live Messenger is not based on the SIMPLE protocol, but rather on the MSNP mentioned earlier. Microsoft's Live Communications Server is based on SIMPLE, and MSNP likely has its basis on SIMPLE as well, but this is not known as MSNP is not available to the public. The major advantage of SIMPLE is its intended integration with voice and video sessions, however the protocol remains in the IETF standardization process and the RFC drafts are complicated and heavyweight (Hildebrand, 2003).

e. Jabber / XMPP

In 1998, in an effort to escape the need for multiple IM accounts, Jeremy Miller created the Jabber protocol. He implemented the first server for IM and presence using Jabber protocols in 1999, and the first public release of the server was in May of 2000 (Saint-Andre and Meijer, 2005).

The Jabber protocol was accepted by the IETF as a standards track protocol in 2004 and re-named as the Extensible Messaging and Presence Protocol. One of the few open standard protocols available for IM, XMPP has approximately 20 widely known server implementations. Of these, slightly more than half are open source, while the remaining are proprietary commercial implementations of the standard. Because XMPP is an open standard, anyone can implement a client or server. There are over 50 client applications written to support XMPP. The majority of these applications are free and many are open source.

In 2005, the company Google announced its entrance into the IM market. Their IM tool, Google Talk, uses XMPP to implement the IM portion of its services. Google's adoption of XMPP added considerable commercial legitimacy to the XMPP protocols. Apple Computers' iChat application is another major commercial implementation of chat and IM that supports XMPP. It is estimated that the XMPP / Jabber network supports over 25 million users (Wikipedia.org, 2006b).

XMPP has been a mandated standard with the DoD IT Standards Registry since November of 2005. XMPP is the only mandated chat and IM standard protocol in the DoD (Barrett, 2006).

2. Extensible Markup Language (XML)

XML is a meta-language, used to define other data languages. It is a subset of the Standard Generalized Markup Language (SGML), and like SGML is used to define mark-up languages. In its December, 1997 press release the World Wide Web Consortium described XML as such,

> XML is primarily intended to meet the requirements of large-scale Web content providers for industry-specific markup, vendor-neutral data exchange, media-independent publishing, one-on-one marketing, workflow management in collaborative authoring environments, and the

> processing of Web documents by intelligent clients. It is also expected to find use in certain metadata applications. XML is fully internationalized for both European and Asian languages, with all conforming processors required to support the Unicode character set in both its UTF-8 and UTF-16 encodings. The language is designed for the quickest possible client-side processing consistent with its primary purpose as an electronic publishing and data interchange format. (W3C Press Release, 1997)

XML has been adopted as intended. It is used widely in support of web content publishing, and its adoption as a data interchange format has also been widespread.

XML is similar to another SGML derivative, HTML, in that it uses tags and nesting to structure data in document form. But XML is not designed to describe any particular type of data. An XML document is an ordered, labeled tree that can be used to model a wide variety of data. The labels describe the logical structure of the data, and if so desired, can describe the nature of the content, but do not necessarily contain semantic information. Figure 2 is a simple example XML document that demonstrates the labeling and nesting of data elements.

```
<Thesis>
        <Title>XML Tactical Chat (XTC) - The application of Extensible Messaging
and Presence Protocol (XMPP) to Command and Control.</Title>
        <Chapter>
                <Number>I.</Number>
                <Name>Introduction</Name>
        </Chapter>
        <Chapter>
                <Number>II.</Number>
                <Name>Background and Related Work</Name>
        </Chapter>
        <Chapter>
                <Number>III.</Number>
                <Name>Military Chat</Name>
</Chapter>
```

Figure 2. XML Documents label data and structure the data into a nested tree.

While an XML document itself is simply a container for data, the utility in XML lies in the multiple technologies that support XML documents. There are XML tools that allow for the parsing, validating, processing, and transforming the document and its data. Document Type Definitions and XML Schema are used to define the XML document

type and to validate an instance of an XML document for correctness. XPath and XQuery allow for the query and access of XML document contained data, and the rapid insertion or modification of XML data in a document. Cascading Style Sheets and Extensible Stylesheet Language (XSL) are used to display or transform XML documents. These and other XML related technologies form a powerful toolset with which XML documents and data can be applied to solve many computing and information technology problems.

XML namespaces provide a solution to the problem of personalized XML vocabularies. XML, being only a syntax for language definition, does not restrict the use of labels for data. If an XML document intends to rely on multiple document types or languages, it is possible for a particular label to be overloaded across those languages or document types. An XML namespace is a uniquely named category, denoted by a Uniform Resource Identifier (URI), which allows for the disambiguation of overloaded tags with respect to document processing. All data that is to be processed according to a particular document type is effectively prefixed with its associated namespace. XML namespaces add the value of modularity to the XML toolset.

C. RELATED WORK

The related work presented in this section focuses on areas that expand the use of XMPP beyond chat and instant messaging support. Some of these areas involve making XMPP more efficient or taking advantage of XMPP's basis in XML. Others are areas of research that involve using XMPP as a general purpose XML communication backbone.

1. Distributed Interactive Simulation (DIS)

Distributed Interactive Simulation (DIS) is a set of Institute of Electrical and Electronics Engineers standard protocols for conducting real-time war-gaming and simulation across multiple platforms. It is used extensively in the realm of military simulation. Recently, efforts have been made to create an XML representation of the DIS data packets. Expressing DIS data as XML avails the virtual interface to be accessed by XML developers and enables the querying, transforming, storage, and web publishing of the data as with any other XML data (McGregor, Brutzman, Armold, and Blais, 2006). Figure 3 provides an example of a DIS-XML message.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<DIS>
    <EntityStatePdu capabilities="0" entityAppearance="0"
                    forceID="0" numberOfArticulationParameters="0">
        <PduHeader pduType="1" protocolFamily="1" timestamp="0"/>
        <EntityID/>
        <Entity/>
        <AlternativeEntity/>
        <EntityLinearVelocity/>
        <EntityLocation x="1.0" y="2.0" z="3.0"/>
        <EntityOrientation/>
        <DeadReckoningParameters
                otherParameters="00000000000000000000000000000000">
            <EntityLinearAcceleration/>
            <EntityAngularVelocity/>
        </DeadReckoningParameters>
        <EntityMarking characterSet="0" characters="00000000000000000000"/>
    </EntityStatePdu>
</DIS>
```

Figure 3. DIS PDUs can be expressed as XML Messages.

DIS data generally relies on multicasting as its transmission method within a Local Area Network. The Internet, using the point to point architecture of Transmission Control Protocol (TCP) and User Datagram Protocol (UDP), has become the dominant network on the physical infrastructure. Most routers that service the Internet are not configured to route multicast packets.

DIS-XML conversion is not yet complete, but the arena presents an interesting avenue in demonstrating the potential of XMPP as a general purpose streaming XML protocol. Using DIS-XML in lieu of native DIS does reduce CPU performance and is more consumptive of bandwidth, however, DIS-XML has been demonstrated as usable in at least two examples: the Autonomous Underwater Workbench and the L-3 demonstration at IITSEC 2005 (McGregor et al., 2006). The primary benefit of DIS-XML is achieving multicast-style capability through firewalls and across the Internet, otherwise not possible without the use of bridges. Continuing work in Efficient XML interchange (EXI) is likely to improve processor and network performance of DIS-XML.

2. Joint Consultation Command and Control Information Exchange Data Model (JC3IEDM)

The Joint Consultation Command and Control Information Exchange Data Model is an extension of the Multi-Lateral Interoperability Program's (MIP) Command and Control Information Exchange Data Model (C2IEDM). MIP is a multinational organization whose goal is:

> The aim of the MIP is to achieve international interoperability of Command and Control Information Systems (C2IS) at all levels from corps to the lowest appropriate level, in order to support multinational, combined and joint operations and the advancement of digitization in the international arena, including NATO(Multilateral Interoperability Programme, 2006).

JC3IEDM is the most current data model of MIP and NATO. It is the common language by which Command and Control systems, across military services, would exchange information. Originally, C2IEDM consisted solely of a collection of relational database schemas. These schemas have since been transformed to XML schemas, and comprise a rich set of messages and documents that sufficiently cover the military command and control domain, particularly as applied to land operations. JC3IEDM XML messages are valuable in that they can be purposed as information exchange objects themselves or used to populate and update a JC3IEDM database servicing situation awareness and other command and control requirements. The marriage of JC3IEDM XML messages and XMPP provides one of the most promising extensions of Chat/IM for military purposes.

3. XML Binary Serialization

a. Introduction

The concept of binary characterization of XML existed since the XML standards were created, but the community did not begin any rigorous approach to standardizing the process until 2003. In September of 2003, the World Wide Web Consortium (W3C) formed the Workshop on Binary Interchange of XML Information Item Sets. The three day workshop was formed to collect information about whether a W3C standard for XML binary characterization was necessary, what work had been done in the area up to that point, and what use-cases exist for binary XML (Pericas-Geertsen, 2003). The result of this workshop was that the area was important enough to form a W3C Working Group on the subject. Formed in April of 2004, the XML Binary Characterization Working Group concluded in March of 2005, having delivered the four publications it was tasked to produce: Properties, Use Cases, Measurements, and Characterization. Table 1 lists the 18 use cases published by the working group (Cokus and Pericas-Geertsen, 2005).

W3C XML Binary Use Cases
1. Metadata in Broadcast Systems
2. Floating Point Arrays in the Energy Industry
3. X3D Graphics Model Compression, Serialization, and Transmission
4. Web Services for Small Devices
5. Web Services within the Enterprise
6. Electronic Documents
7. FIXML in the Securities Industry
8. Multimedia XML Documents for Mobile Handsets
9. Intra/Inter Business Communication
10. XMPP Instant Messaging Compression
11. XML Documents in Persistent Store
12. Business and Knowledge Processing
13. XML Content-based Routing and Publish Subscription
14. Web Services Routing
15. Military Information Interoperability
16. SyncML for Data Synchronization
17. Sensor Processing and Communication
18. Supercomputing and Grid Processing

Table 1. Use cases assembled by XBC for a binary XML encoding.

A number of the uses cases identified have potential for military application and have relevance to this thesis. Use case 15 is itself a description of need for efficient interoperable data within the military, and use case 10 is the application of binary characterization to XMPP streams. Each will be examined in greater detail.

b. Military Information Interoperability

The essence of this use case is that while the U.S. military and its allies have widely adopted XML technologies as a solution for information interoperability, there exists a rift between those information systems that have the resources to process and store text-based XML and those that do not. The majority of tactical information systems, being radio/wireless based, lack the necessary bandwidth and/or local memory and storage to handle and process raw XML. There is, therefore, great need in the military for a standard, by which an efficient XML interchange solution may be implemented across all information systems. Doing so would serve to greatly increase command and control capability and information interoperability across the entire breadth of the military networks. The stated desire of the DoD in the Use Cases documentation is:

> The DoD would like a single binary XML standard that works well for the diverse range of data, systems and scenarios specified above as opposed to an incompatible set of binary XML standards optimized for vertical industries. We would also like a binary XML standard that leverages schema information when it is available to improve performance, but does not depend on its existence or accuracy to work. We have sponsored commercial research and development that demonstrates that what we want is both possible and practical ... (Cokus and Pericas-Geertsen, 2005).

c. XMPP Instant Messaging Compression

The analysis of this use case found several potential advantages in applying binary compression to XMPP message streams. First, the addressing information for each message is contained in well-known XML nodes, thus, a binary characterization that could be parsed more efficiently than text XML would allow for faster routing of XMPP messages. Second, compact binary formatted XMPP messages will use less bandwidth than uncompressed XML XMPP stanzas, improving suitability for bandwidth constrained networks. Lastly, there are many XMPP messages that are slight modifications of previous messages, an updated presence message for example, and a more efficient method of updating these messages would prove beneficial to an XMPP network. One final item presented in this use-case is that any binary characterization must be transcodable back to the original text XML format in order to maintain conformance with the XMPP RFC specifications (Cokus and Pericas-Geertsen, 2005).

d. Efficient XML Interchange (EXI)

Upon completion of the XML Binary Characterization Working Group, a second working group was chartered; The Efficient XML Interchange Working Group. Published in November of 2005, the EXI working group charter states that the mission of the working group is to develop a format for the efficient interchange of the XML information set based on the findings of the XML Binary Characterization Working Group. The goals of the working groups according to the charter are:

1. Fulfill the design goals of XML with the following exceptions:
    a. The interchange format must be compatible with the XML Information Set instead of being “compatible with SGML” (XML goal 3);

b. For performance reasons, the format is not required to be "human–legible and reasonably clear" (XML goal 6);

c. Terseness in efficient interchange is important (XML Goal 10).

2. Address all requirements and use cases from the XML Binary Characterization Working Group;

3. Maintaining the existing interoperability between XML applications, as well as XML specifications;

4. Establish sufficient confidence in the proposed format, in particular establishing confidence that the performance gains are significant, and the potential for disruption to existing processors is small…(Goldman, Berjon, and Bournez, 2005)

The charter expires on December 31, 2007. The forthcoming standard for binary characterization and interchange of XML will be an important aspect of future XMPP based systems. Implementation of the standard in the DoD will avail the interoperability of chat/IM, and potentially many other command and control systems to the edges of the military network.

4. XML Storage/Search

a. Introduction

Since XML's emergence as a versatile and powerful data storage container, there has been the need for, and development in, the storage and access of XML based data. Today, there are three major means of storing, querying and retrieving XML data: flat file storage, insertion into a relational database, and implementing a native XML database (Bouret, 1999). The suitability of these methods depends largely on the type of XML data being stored and accessed. XML documents can be generally classified as either data-centric or document-centric. Data-centric XML, typically for machine use, uses XML formatting strictly as a data container, placing no meaning on the structure of the document itself. Document-centric XML, typically for human use, relies on the structure of the document itself as well as the data contained in the document. XHTML web content is an example of document-centric XML(Bouret, 1999).

b. Flat File Storage

Flat file storage is generally suitable only for small scale applications. It presents some complication in querying as there is no distinction between meta-data and content data. One solution to this problem is the use of XML query languages such as

XPath or XQuery or even XSLT. These languages, particularly XQuery, allow for the traversal and access to XML data. An alternative solution to this problem is to use an Information Retrieval (IR) Library such as Lucene, to index XML documents and allow for search over data content. Lucene and other IR libraries are not specific to XML use, rather are an abstraction to allow for indexing and search capabilities to be inserted into any application. Such IR libraries have potential for integration into military command and control systems by inserting search and retrieval tools into existing applications.

c. Relational Databases

The second method of XML storage and retrieval is the use of relational databases. These databases generally do not use the structure of XML as the basis for storage. Some databases store XML documents as Binary Large Objects (BLOBS) or Character Large Objects (CLOBS) and provide XML aware indexing to data contained in the document (Bouret, 1999). Others create a mapping from the XML document schema to the relational schema, a process called shredding. There are some large commercial database systems that to provide native XML data handling as an extension to their relational database format.

d. Native XML Databases

The final method of XML storage is the use of a native XML database. Native XML databases provide a number of advantages over mapping to relations or indexing stored large objects. These benefits are related to the fact that the data storage model is based on the same format as the XML document. This can be leveraged to allow for effective query on document-centric XML that would prove difficult using relational database storage (Bouret, 1999).

e. Coalition Secure Management and Operations System (COSMOS)

Coalition Secure Management and Operations System (COSMOS) is an Advanced Concept Technology Demonstration (ACTD) managed out of the Office of the Secretary of Defense for Acquisitions, Technology, and Logistics (OSD-ATL). The objectives of the COSMOS demonstration are listed below (Burns, 2006).

- Enable unambiguous protected sharing of C2 info for more rapid and decisive ops among coalition partners on a single multinational information sharing network

- Enable protected lateral comms across a collapsed environment
- Adoption of US and coalition data model
- Enable machine-to-machine exchange of C2 info between MIP-compliant systems and applications
- Reduce number of coalition networks required to support Coalition Task Force ops within a theater of operations
- Enable coalition members to share information with each other based upon roles
- Smart agents to further reduce direct human handling
- Enable all coalition partners to engage with, or disengage from, the coalition without disrupting the information sharing capability of the network
- Single secure MNIS network for a broad range of coalition partners with MIP-compliant C2 systems and applications

COSMOS is related to XTC in that the machine-to-machine information exchange objective of COSMOS is to be achieved using the C2IEDM data model, which is a subset of the JC3IEDM. XTC research into using XMPP to route XML-expressed JC3IEDM data with Navy Exercise Trident Warrior 2006, was sponsored by the COSMOS program. This work is presented in Chapters VI and VII.

f. Conclusion

The XML basis of XMPP systems creates a decision point on how to store and access the collected chat/IM data. Flat files, relational databases and native XML databases, along with XQuery and other query languages, all allow for access to XML stored data. This thesis provides, as an example, the use of a native XML database for storage and query of chat conversation logs.

D. SUMMARY

This chapter presented some background information to the key technologies of XMPP, Instant Messaging and XML. Additionally, related topics were presented in the areas of DIS-XML, JC3IEDM, XML binary compression and XML storage and search.

THIS PAGE INTENTIONALLY LEFT BLANK

III. MILITARY CHAT

A. INTRODUCTION

Chat and IM technology have been widely adopted across the military services. Up to now, military organizations have largely used IRC or one of the large commercial IM tools discussed in Chapter II. These tools, however, were not designed for military use and fail to meet many of the requirements of military information systems. For those chat and IM tools built on proprietary protocols, it is not expected they will be easily adapted and enhanced for military use either. Much development in commercial chat and IM is in the area of user interface and enhancing the user's experience. Military chat needs are far more utilitarian, requiring information security and assurance, account management, access control, and other administrative capabilities. The utility of chat and IM is significant, but there remains a large problem space in the field of defining and developing chat and IM tools for military purposes.

B. CHAT RESEARCH

Given the amount of chat and IM use in the military, there is a seemingly small amount of research into the area. Captain Bryan Eovito, USMC, while a Master's Degree student at the Naval Postgraduate School, wrote his thesis on joint chat requirements drawn from usage patterns. Captain Eovito sought to collect and analyze existing chat research, finding little published material. The unpublished e-mails, internal reports, and publications used to comprise his knowledge base did not represent a comprehensive analysis of military chat use and needs. Eovito did identify three organizations who have conducted recent research into military chat: the Center for Naval Analyses (CNA), the Pacific Science and Engineering Group, and the Massachusetts Institute of Technology (MIT) (Eovito, 2006).

The CNA has conducted research on chat and analyzed operational chat use across the Navy and through the use of experiments as well. In 2002, this group conducted a survey of Navy chat status and presented a framework for an enterprise-wide Navy chat solution. In 2003, follow up recommendations for at sea fleet chat support were presented. In 2004 and 2005, as part of Exercise Trident Warrior 2004, this group studied distributed chat architecture in support of military operations.(Eovito, 2006)

The Pacific Science and Engineering Group's research focused on chat use in support of Operation Enduring Freedom and Operation Iraqi Freedom. This research surveyed chat users during this period to assess use patterns and requirements. Additional research by this group is in the area of usability and Human Systems Interface (HSI). The MIT group's research also focused on interfaces in support of chat communications (Eovito, 2006).

Additional chat related research, focused on XMPP based solutions, is discussed in detail in Chapters VI and VII.

C. CHAT REQUIREMENTS

1. XTC Technical Report Requirements

The XTC Technical Report, released by the MOVES Institute, Naval Postgraduate School, in Monterey, CA presents recommended requirements for military chat. It categorizes these requirements into three groups: tactical requirements, technical requirements, and administrative requirements. Tactical requirements are defined as when and why chat is being conducted in the military environment. Technical requirements are defined as issues related to the data formats and application design for military chat. Administrative requirements are defined as those rules concerning chat conduct, information assurance policies, and user level implementation of chat (Brutzman et al., 2004)

Figures 4, 5, and 6 present these recommended tactical, technical, and administrative requirements.

- Support command and control to include ongoing dialog as well as situation reports, execution checklist milestones, and casualty reports.
- Support operational planning at the micro and macro levels for both upcoming and real-time event scheduling and coordination
- Support coordination efforts for administrative support, logistics, technical support, and other day-to-day requirements.
- Log all chats so that valuable information is preserved for search and ready analysis.
- XML-ize chat to reduce the effort required to extract information
- Use and open source solution that provides information assurance and is extensible for future requirements.

Figure 4. The recommended tactical requirements for XML-based tactical chat describe functional chat needs. Taken from (Brutzman et al., 2004).

- Mark up chat using standardized XML
- Process plain prose, message-text format (MTF), BGH Tactical Markup Language (BGH TML), HTML, and BGH ATTCS reference model messages.
- Use XSLT templates for arbitrary addition, deletion, and modification of available messages as required.
- Validate messages against a schema
- Employ BGH elements compatible with the BGH architecture.
- Accept and store MIME types and URLs
- Implement an SMS bridge for cell-phone text messaging
- Incorporate data mining support into application
- Define audio, video, and whiteboarding schema
- Implement an asynchronous system
- Implement a thin client for single Web browser use
- Maintain interoperability among clients and systems
- Maintain firewall-policy compatibility
- Choose open-source code to preclude security loopholes, reduce cost, and encourage broad development.

Figure 5. The recommended technical requirements address data format and application design issues for military chat. Taken from (Brutzman et al., 2004).

- Define administrator, moderator, author, and participant roles
- Define chat-room permissions
- Limit post content to relevant information
- Provide an interface for scheduling chat
- Designate areas of special interest for chat
- DoD Information Technology Security Certification and Accreditation Process (DITSCAP) compliant

Figure 6. The recommended administrative requirements address chat user conduct and accreditation needs for military chat. Taken from (Brutzman et al., 2004).

2. Requirements Documents

Most DoD requirements documentation for chat and IM are contained within the requirements for collaboration applications. For example, the Deployable Joint Command and Control system (DJC2) is a set of horizontally and vertically integrated command and control applications that combine to create a Collaborative Information Environment to support Joint Operations (USJFCOM, 2003). The baseline requirements document for DJC2, describes requirements in many of the same areas as was presented as use pattern requirements for chat: information assurance and security, scalability, administration and infrastructure, and information and application sharing are all addressed. Text-chat is addressed as an embedded function or application within the collaboration system. The functional requirements for text chat features within DJC2 are listed in Figure 7 (USJFCOM, 2003).

- One-to-One Communication - DJC2 shall provide the capability to privately text chat between individual workstations.
- One-to-Many Communication - DJC2 shall provide the capability for an individual participant to conduct text chat sessions with multiple selected participants.
- Many-to-Many Communication - DJC2 shall provide the capability to conduct simultaneous text chat sessions between multiple users.
- Save/Archive/Time Stamp/Retrieve - DJC2 shall provide users and administrators the capability to save, archive, time-stamp and retrieve text chat sessions.
- Foreign Language Text Translation - DJC2 shall provide the capability to perform foreign language translation for text and data.

Figure 7. The Deployed Joint Command and Control system has requirements for incorporated text chat.

3. Requirements from Usage Patterns

In addition to aggregating research on military chat use, Eovito examined the contents of hundreds of operational and exercise after-action reports and lessons learned and conducted additional study on tactical level chat use, surveying 58 military chat users, from various nations. Through this analysis Eovito identified consolidated user requirements for chat and IM in four areas: functional requirements, information assurance requirements, scalability requirements, and interoperability requirements. Table 2 lists the identified requirements in the functional requirement area. Functional requirements address those features of the chat client interface. Most are features derived from the chat and instant messaging domain, while others are drawn from military information handling. The majority of these requirements can be met by commercially or open-source available solutions, though it is unlikely that a single solution addresses all 30 requirements.

Consolidated Functional Requirements
Functional Requirements
(* denotes a core requirement)
1. Participate in Multiple Concurrent Chat Sessions*
2. Display Each Chat Session as Separate Window
3. Persistent Rooms and Transitory Rooms*
4. Room Access Configurable by Users
5. Automatic Reconnect and Rejoin Rooms*
6. Thread Population/Repopulation*
7. Private Chat "Whisper"*
8. One-to-One IM (P2P)
9. Off-line Messaging
10. User Configured System Alerts
11. Suppress System Event Messages
12. Text Copying*
13. Text Entering*
14. Text Display*
15. Text Retention in Workspace*
16. Hyperlinks
17. Foreign Language Text Translation
18. File Transfer
19. Portal Capable
20. Web Client
21. Presence Awareness/Active Directory*
22. Naming Conventions Identify Functional Position*
23. Multiple Naming Conventions
24. Multiple User Types
25. Distribution Group Mgmt System for Users
26. Date/Time Stamp*
27. Chat Logging*
28. User Access to Chat Logs*
30. Interrupt Sessions

Table 2. Consolidated Functional Requirements for Chat and IM. (From: Eovito, 2006).

The Information Assurance requirements are presented in Table 3. These requirements address the information security and assurance needs for chat use in the military. Any future technology for military chat and IM must be easily extended and modified to support these information security requirements. While such extension is possible with the use of open standards, it is highly unlikely that a proprietary standard, such as that used by AOL, Yahoo, or Microsoft, would be modified to implement military specific requirements.

Consolidated Information Assurance Requirements
Information Assurance Requirements
1. Login and User Authentication
2. Access Control
3. User Authentication by Active Directory
4. Unique ID for all users worldwide
5. PKI Enabled (DoD Common Access Card)
6. Provide Encryption
7. Network Security Tools
8. Cross Security Domain Functionality
9. Multi-Level Security Operation

Table 3. Consolidated Information Assurance Requirements for Chat and IM. (From: (Eovito, 2006).

The scalability requirements for military chat and IM are presented in Table 4. These requirements address the need for chat solutions to retain suitability across both robust and austere networks, and address concurrent user related issues. Many large chat and IM systems can easily support the user load required by the military. However, there are only a few solutions that can support distributed enterprise architecture. To meet all of the scalability requirements, an extensible chat and IM solution will be required.

Consolidated Scalability Requirements
Scalability Requirements
1. Austere Network Operation
2. Low Overhead Login Process
3. Use Client without Server
4. Distributed Architecture
5. Number of Concurrent Chat Sessions
6. Number of Concurrent Users
7. Specified Quality of Service

Table 4. Consolidated Scalability Requirements for Chat and IM. (From: Eovito, 2006).

Table 5 lists the interoperability requirements for military chat and IM. These, too, are not addressed by currently implemented solutions and are unlikely to be met by any single commercially available product. It is important to note that many of the functional, information assurance, and scalability requirement sets will vary between services and organizations. The interoperability requirements must be met, even as disparate feature sets for chat and IM are implemented across the DoD.

Consolidated Interoperability Requirements
Interoperability Requirements
1. DoD Standards
2. Open Standard
3. Multi-Platform Clients
4. Interoperate with Existing Collaboration Systems
5. Interoperate With Office Automation Tools

Table 5. Consolidated Interoperability Requirements for Chat and IM. (From: Eovito, 2006).

D. SUMMARY

This chapter discussed the state of research on military chat use and presented a list of chat and IM requirements derived from this research. At the time of writing, none of the services has formalized the requirement quantities, feature sets, and performance measures for chat and IM. It is certain, however, that the needs of the services and organizational units will vary across the DoD. Future military chat and IM solutions must be able to flexibly support these variations, while maintaining chat interoperability between the different implementations. Of the current chat and IM solutions today, XMPP presents the greatest potential for meeting these extensibility and interoperability requirements.

IV. EXTENSIBLE MESSAGING AND PRESENCE PROTOCOL (XMPP)

A. INTRODUCTION

This chapter presents the background, specifications, and current implementations and status of the XMPP protocol.

B. HISTORY

XMPP originated in the concept of XML streams, developed by Jeremy Miller in 1998. Originally coined as "Jabber," the protocol became formalized by the IETF under the name Extensible Messaging and Presence Protocol. Though conceived as an open and interoperable solution to multiple instant messaging systems, XMPP is not specifically an IM or Chat protocol. Rather, XMPP has a core specification, RFC 3920, that describes streaming XML in generic terms, along with incorporated security protocols, (Saint-Andre, 2004a) and a second specification, RFC 3921 that describes the instant messaging and presence extensions of the core specification (Saint-Andre and Meijer, 2005).

In addition to the two base IETF approved standards mentioned above, there are three other IETF standards for XMPP: RFC 3922 (Mapping the Extensible Messaging and Presence Protocol (XMPP) to Common Presence and Instant Messaging (CPIM)), RFC 3923 (End-to-End Signing and Object Encryption for the Extensible Messaging and Presence Protocol (XMPP)), and RFC 4622 (Internationalized Resource Identifiers (IRIs) and Uniform Resource Identifiers (URIs) for the Extensible Messaging and Presence Protocol (XMPP)). Additionally, there is a set of extension protocols that are managed by the Jabber Software Foundation (JSF). These extensions are called Jabber Enhancement Proposals, and undergo a standards-tracking process similar to that of the IETF, but managed by the JSF. The JEPs are managed through a process that is defined in the first published JEP, JEP-0001.

C. SPECIFICATION

1. Core

The XMPP core specification is the IETF approved RFC 3920, titled "Extensible Messaging and Presence Protocol (XMPP): Core." The abstract of the publication states that:

> This memo defines the core features of the Extensible Messaging and Presence Protocol (XMPP), a protocol for streaming Extensible Markup Language (XML) elements in order to exchange structured information in close to real time between any two network endpoints. While XMPP provides a generalized, extensible framework for exchanging XML data, it is used mainly for the purpose of building instant messaging and presence applications that meet the requirements of RFC 2779. (Saint-Andre, 2004a)

Interestingly, the last sentence places emphasis on the instant messaging application of XMPP. It is the position of this thesis that equal value lies, in fact, in its provision of a "generalized, extensible framework for exchanging XML data…" (Saint-Andre, 2004a).

a. Jabber Identifiers (JIDs)

All network entities that can communicate using XMPP must have a unique identifying address. These identifiers are called Jabber Identifiers or JID. A valid JID contains a set of ordered elements formed of a domain identifier, node identifier, and resource identifier. Like e-mail addresses, the JID uses the following format: node@domain/resource. Because of the inclusion of the domain portion of the JID, JIDs are generally considered to be unique within a given worldwide network, such as the Internet.

XMPP addressing is well suited for integration into military organizations. The naming structure of the JID can easily be mapped to existing e-mail accounts, such that XMPP address lookup services could be implemented along side the Global Address Lists provided by e-mail exchange servers. Many XMPP server implementations provide user authentication with Windows Active Directory and Lightweight Directory Access Protocol for example. Additionally, an XMPP server network, distributed according to command and control structure would allow the JID inclusion of domain and resource identifiers to become descriptive in their operational context, as well as their network-centric one. This would facilitate storage and retrieval of XMPP messaging based on units, organizations, and individuals without having to add additional meta-data to each message.

b. XMPP Security

XMPP has security requirements written into its core specification. All client sessions require user authentication. Self-account creation is permissible, but can be disabled to allow for strict control of user accounts. Client to server sessions are supported by both Transport Layer Security (TLS) for stream protection between nodes. Simple Authentication and Security Layer (SASL) is used to provide secure authentication of the user. XMPP uses three ports to support all communication: ports 5222/5223 for client to server sessions and port 5269 for server-to-server session. These ports are registered with the Internet Assigned Numbers Authority and provide ready administrative control of XMPP traffic through firewalls. Finally, XMPP has specified handling procedures for BASE64 data, providing additional protections against information leaks and buffer overflow attacks (Saint-Andre, 2004a).

c. XMPP Message Passing

While not required to, XMPP implementations typically use client-server architecture. XMPP users are required to authenticate to an XMPP server in order to send messages. Much like SMPT, XMPP Messages are not sent directly between clients, rather they are sent from the sender's client to the server, passed to a server that supports the receiver's client, and then delivered to the receiver's client. Servers maintain awareness of each other through Domain Name System, and as mentioned above, the server hostname is part of a user's JID. Figure 8 diagrams how XMPP passes message traffic from one client to another.

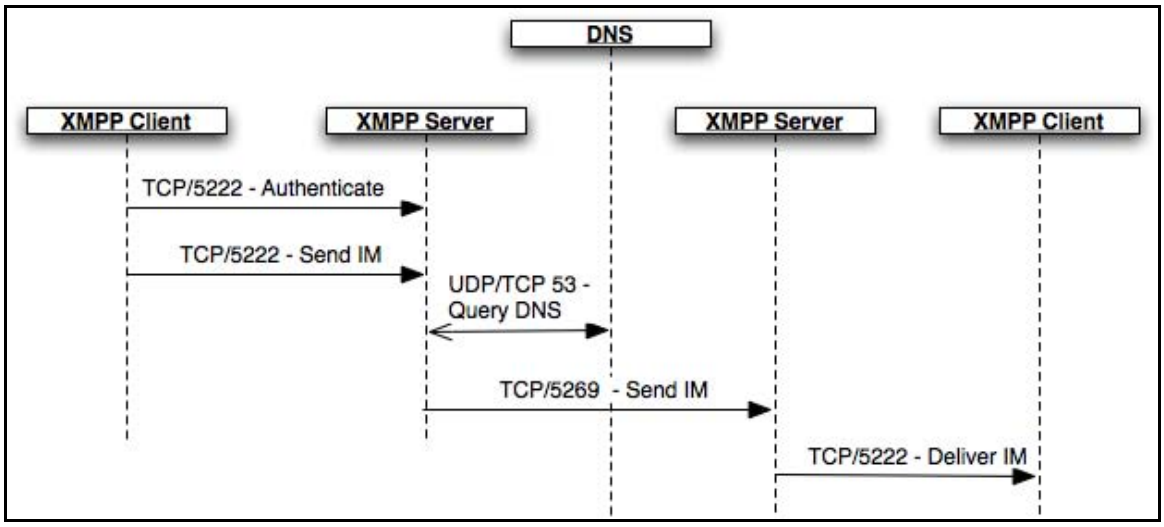

Figure 8. XMPP typically uses client – server architecture. Users authenticate to a server. Messages are sent from client to server to server to client.

d. XML Streams and Stanzas

RFC 3920 defines an XML stream as a container for the exchange of XML elements between any two entities over a network. The opening of a stream is denoted by the <stream> tag along with any appropriate attribute and namespace declarations. Until closed by a </stream> tag, the initiating entity can send any number of XML elements to the recipient. Some of these elements are used to negotiate the session initiation and security procedures, and other elements are XML stanzas that are discrete semantic units of structured XML data (Saint-Andre, 2004a).

Stanzas are direct child elements of the XML stream. The core specification defines three stanzas in an XMPP XML stream: <iq/>, <presence/>, and <message/>. These three elements serve unique purposes. <iq/> elements are Info/Query elements. They operate using a request-response mechanism that enables an entity to make a request of, and receive a response from, another entity. The <presence/> element can be seen as a basic broadcast or "publish-subscribe" mechanism, by which multiple entities receive information about an entity to which they have subscribed. The <message/> stanza contains information that is pushed from one entity to another, similar to the communications that occur in email systems (Saint-Andre, 2004a).

2. Chat/IM

RFC 3920 describes the core specification for XMPP; however, the impetus for creating the standard was the desire for an open standard for instant messaging. As such, a second standard was drafted and approved by the IETF, RFC 3921. RFC 3921 is titled "Extensible Messaging and Presence Protocol (XMPP): Instant Messaging and Presence." It describes the extensions to and application of the XMPP core standard to instant messaging and presence (Saint-Andre, 2004b).

The IETF has defined the requirements that an instant messaging and presence system must be able to meet. These are defined by the following use cases: (Saint-Andre, 2004b)

- Exchange messages with other users
- Exchange presence information with other users
- Manage subscriptions to and from other users
- Manage items in a contact list (in XMPP this is called a "roster")
- Block communications to or from specific other users

RFC 3921 details how these use cases are met within the XMPP construct and in compliance with the Core standard. RFC 3921 describes the mandatory and recommended syntax of the <iq/>, <presence/>, and <message/> stanzas. The standard also establishes rules for exchanging messages of each stanza type. Additionally, the RFC describes stanza handling by XMPP servers, roster management, and internationalization and security concerns.

Each of the three XML stanzas has specified child element content that supports the instant messaging function. It is recommended that <message/> stanzas contain a type attribute that describes the nature of the message stanza. If used, the type attribute must be one of the five listed below: (Saint-Andre, 2004b)

- chat - a one-to-one text message in the context of a conversation
- error – describes an error resulting from a previous message sent
- groupchat - a one-to-many conversation text message in the context of a group conversation

- headline - a message generated by an automated service that delivers or broadcasts content
- normal – a single message sent outside the context of one-to-one or many-to-many conversation

A message stanza may contain any properly namespace qualified child elements, but there are three specified child elements that may be used without namespace qualification. These are the <subject/>, <body/>, and <thread/> elements. The first two are to contain human readable XML and serve the purpose to describe the topic of the message and the textual content of the message respectively. <thread/> elements are non-human readable XML that is used to specify an identifier that can be used to track a conversation or instant messaging session. (Saint-Andre, 2004b)

Presence stanzas also may contain a type attribute of one of the following types: (Saint-Andre, 2004b)

- unavailable - Signals that the entity is no longer available for communication.
- subscribe - The sender wishes to subscribe to the recipient's presence.
- subscribed - The sender has allowed the recipient to receive their presence.
- unsubscribe - The sender is unsubscribing from another entity's presence.
- unsubscribed - The subscription request has been denied or a previously-granted subscription has been cancelled.
- probe - A request for an entity's current presence; SHOULD be generated only by a server on behalf of a user.
- error - An error has occurred regarding processing or delivery of a previously-sent presence stanza.

Presence stanzas, too, may contain any properly namespaced child elements, and also have optional child elements that do not require namespacing. These are the <show/>, <status/>, and <priority/> elements. <show/> specifies the availability status of the entity or resource in non-human readable XML, <status/> is a natural language description of the availability status (i.e. out to lunch), and <priority/> is a numeric priority level indicator for server handling purposes.

<iq/> stanzas also have specified extensions in the instant messaging and presence application of XMPP, but unlike the message and presence stanzas, these extensions are implemented with extended namespaces. The two extensions applied allow for roster

management and for communication blocking. These are not described in detail here. Interested readers should refer to IETF RFC 3921 for further information.

3. Jabber Enhancement Proposals (JEPs)

Jabber Enhancement Proposals are extensions of the XMPP standard, managed by the Jabber Software Foundation. There are five types of JEPs: Historical, Informational, Procedural, Standards Track, and Humorous.

Historical JEPs describe a practice widely in use by the XMPP and Jabber community, but are not official protocols or practices. The practice described by Historical JEPs may become standardized or may be superseded by a different standardized specification of the practice.

Informational JEPs typically describe best practices and protocol usage but again are not official standards nor are they in the standards track process.

Procedural JEPs are descriptions of the organizational procedures of the Jabber Software Foundation.

Standards Track JEPs may be considered standardized extensions of the XMPP protocol. This is not to say that they are IETF standards, as they are not. However, Standards Track JEPs are managed by the JSF through an open process and provide a platform for XMPP development to expand and mature. Standards Track JEPs have two types, Final JEPs and Draft JEPs. Final JEPs have been approved as final standards by the JSF and are no longer subject to modification. They are considered stable for implementation and deployment. Draft JEPs are also considered safe for deployment; however, they are still under consideration by the Jabber Council and are subject to modification prior to achieving Final JEP status.

In addition to the draft and final JEPs, there is a set of Experimental JEPs that are managed as the related practice or protocol is developed further. There are some common IM application functions, such as Multi-User Chat that are in the JEP domain when implemented over XMPP. Additionally, there are a number of JEPs that have great potential for application in the military Command and Control context.

a. Multi-User Chat (MUC)

IRC is the group chat protocol most widely used in the U.S. Navy. JEP-0045: Multi-User Chat describes the standard implementation of group chat over XMPP. XMPP MUC is basically implemented according to these rules: (Saint-Andre, 2005)

- Each room is identified as <room@service> (e.g., <jdev@conference.jabber.org>), where "room" is the name of the room and "service" is the hostname at which the multi-user chat service is running.
- Each occupant in a room is identified as <room@service/nick>, where "nick" is the room nickname of the occupant as specified on entering the room or subsequently changed during the occupant's visit.
- A user enters a room (i.e., becomes an occupant) by sending presence to <room@service/nick>.
- Messages sent within multi-user chat rooms are of a special type "groupchat" and are addressed to the room itself (room@service), then reflected to all occupants.
- An occupant can change his or her room nickname and availability status within the room by sending presence information to <room@service/newnick>.
- An occupant exits a room by sending presence of type "unavailable" to its current <room@service/nick>.

The MUC JEP addresses a large set of administrative chat room management functions. Table 6 lists the functions and features to be implemented by XMPP MUC.

Features and Functions of XMPP Multi-User Chat	
1.	Native conversation logging (no in-room bot required)
2.	Enabling users to request membership in a room
3.	Enabling occupants to view an occupant's full JID in a non-anonymous room
4.	Enabling moderators to view an occupant's full JID in a semi-anonymous room
5.	Allowing only moderators to change the room subject
6.	Enabling moderators to kick participants and visitors from the room
7.	Enabling moderators to grant and revoke voice (i.e., the privilege to speak) in a moderated room, and to manage the voice list
8.	Enabling admins to grant and revoke moderator privileges, and to manage the moderator list
9.	Enabling admins to ban users from the room, and to manage the ban list
10.	Enabling admins to grant and revoke membership privileges, and to manage the member list for a members-only room
11.	Enabling owners to limit the number of occupants
12.	Enabling owners to specify other owners
13.	Enabling owners to grant and revoke administrative privileges, and to manage the admin list
14.	Enabling owners to destroy the room

Table 6. Features and functions of Multi-User Chat identified by JEP-0045: Multi-User Chat (Jabber Software Foundation JEP-0045)

In addition to these basic features regarding Multi-User Chat rooms, the JEP also identifies various room types. These features are particularly applicable to military adaptation for the purposes of information security. The concerns identified by Captain Bryan Eovito in his research of military chat requirements regarding un-moderated access to chat rooms by military personnel can be addressed by the features listed in Table 7.

Multi-User Chat Room Types specified by JEP-0045.	
1.	public or hidden
2.	persistent or temporary
3.	password-protected or unsecured
4.	members-only or open
5.	moderated or unmoderated
6.	non-anonymous or semi-anonymous

Table 7. Multi-User Chat Room types specified by JEP-0045. (From: Jabber Software Foundation JEP-0045)

The functions and features provided by JEP-0045 provide a rich set of tools for configuration management of MUC rooms that address many of the information management requirements held by the military. These tools allow for DoD enterprise architecture solutions for Chat/IM that can be built based on the requirements of existing command and control structure and that addresses information security concerns.

b. Other Jabber Enhancement Proposals with Military Application

There are a large number of JEPs, covering a wide area of functional extensions to XMPP. A large number of these, some in Draft or Final form, others in Experimental status, offer great applicability to military use cases and afford opportunity to expand the capabilities of Chat/IM in support of Command and Control.

(1) JEP-0060: Publish-Subscribe. Publish-Subscribe (pub-sub) is a content dissemination model in which a publisher releases content and either the content or a notification of it is sent out to a group of subscribers. Some of the existing needs for publish-subscribe are news feeds and content syndication, extended presence, workflow systems, network management systems, profile management, event notification. Additionally, there are a number of military functions that would be well served by disseminating information is this manner, to include issuance of orders, intelligence dissemination, and even position/location data (Millard, Saint-Andre, and Meijer, 2005).

(2) JEP-0080: User Geolocation. As mentioned in the section about Publish-Subscribe, user geolocation is a use case for pub-sub. JEP-0080 defines a format for capturing data about an entity's geographical location. The JEP states that: (Hildebrand and Saint-Andre, 2004)

> This JEP defines a format for capturing data about an entity's geographical location (geoloc). The namespace defined herein is intended to provide a semi-structured format for describing a geographical location that may change fairly frequently, where the geoloc information is provided as Global Positioning System (GPS) coordinates. Potential uses for this approach include:
>
> - Publishing geoloc information to a set of subscribers.
> - Querying another entity for its geoloc.
> - Sending geoloc information to another entity.
> - Attaching geoloc information to presence.

Although this JEP defines the GPS data in terms of latitude and longitude, it would be easy to create an XML schema that describes User Geolocation messages using the Military Grid Reference System. Though this area requires significant further research, the use of the publish-subscribe model for position-location information handling in the military holds significant potential.

(3) JEP-0124: HTTP Binding. This JEP is of particular interest to the XML Tactical Chat concept. Because so much of the military's vital operational activity is supported by austere wireless networks, the persistent TCP connections on which Core specified XMPP is bound are not viable. This JEP provides a standard mechanism for network nodes whose connectivity is intermittent or whose node devices are limited in capability. If a Chat/IM architecture is to be applied to all levels of the command and control structure, the solution must gracefully transition across bandwidth constrained austere networks and separate robust infrastructure. This JEP provides a standard for doing so with XMPP (Paterson, Saint-Andre, and Smith, 2006).

(4) JEP-0138: Stream Compression. As discussed in Chapter II, one of the detractors of XML is its lack of space efficiency as a data container. Because any chat and IM solution that will extend across all operational levels must operate on low bandwidth networks, the need for reduced resource consumption of XML streams is

paramount. This JEP discusses two compression standards, ZLIB and LZW, that are an alternative to the compression provided by the Transport Layer Security (TLS) specified by the XMPP Core standard. The emergence of a standard for XML binary serialization from the W3C EXI Working Group will provide a potentially superior alternative to XML text stream compression for XMPP (Paterson et al., 2006).

(5) JEP-0116: Encrypted Sessions. Information Assurance and Security are fundamental requirements in any military communications systems. Though XMPP is designed to establish secure sessions between servers, XMPP was not designed with end-to-end object encryption in mind. End to end encryption of XMPP traffic eliminates any vulnerabilities resulting from server access to the message traffic, by encypting all payload until it is delivered to the receiving client. RFC 3923: End-to-End Signing and Object Encryption for the Extensible Messaging and Presence Protocol (XMPP) and this JEP aim to create standards for this function. Clearly, for any programmatic adoption of XMPP to take place in the DoD, methods for securing XMPP communications must be available. JEP-0116 is in experimental stage, but indicates that the need for end-to-end encryption is valid and will see future address (Paterson et al., 2006).

(6) JEP-0009: Jabber-RPC. Remote Procedure Calls (RPC) are key methods in combining functionality of multiple programs or databases. They are used extensively to create interoperability across platforms and languages. This JEP defines a standard method for implementing XML-RPC across an XMPP network. With the number of stand-alone military information and command and control systems in operation, this JEP standardizes a common method of integration and interoperability of these systems into an XMPP communications network.

(7) JEP-0072: SOAP over XMPP. Simple Object Access Protocol, is an XML definition that describes a method for exchanging messages across platforms and languages. In this capacity it is similar to XML-RPC, and in many ways, XML-RPC and SOAP are competing technologies over the same function. This JEP defines how SOAP may be implemented over XMPP. This JEP and JEP-0009 allow for flexibility in developing application bridging solutions over XMPP. Of additional note is the fact that

XMPP supports both synchronous and asynchronous communications, making it a more comprehensive XML-RPC/SOAP delivery protocol than the typically used HTTP and SMTP.

(8) JEP-0171: Language Translation. This JEP describes a method for sending and receiving text chat messages with translated text. It also provides a method for a client to ask a server or service to translate a text chat message. This functionality has clear uses when operating in a multinational/coalition environment.

D. SUMMARY

With acceptance as an IETF standard and a rich set of open-standard managed enhancements, XMPP is well positioned to expand and mature as a communications technology. XMPP has been widely implemented as a Chat/IM application, particularly as an enterprise solution for businesses and other organizations. Additionally, XML-streams as a generalized communication layer has allowed for a variety of presence-enabled applications Already, XMPP has been implemented in support of foreign exchange banking, content syndication, network management, fleet vehicle tracking, web-services integration, and distributed whiteboarding (Saint-Andre and Meijer, 2005). It is foreseeable that the uses for XMPP will expand and integrate.

By meeting the needs of military chat and instant messaging and allowing for extension into other areas of command and control, XMPP holds great potential for integration into the command and control systems of the military.

THIS PAGE INTENTIONALLY LEFT BLANK

V. XMPP DEPLOYMENT AND IMPLEMENTATION

A. INTRODUCTION

Because of its open-standards nature, XMPP has a large number of implementations in multiple codebases and using a variety of licenses. As described in Chapter II, there are several implementations of XMPP servers and many client applications for XMPP exist. This chapter looks into the state of XMPP development and examines a few of these implementations to demonstrate how an XMPP-based chat system may be employed as a command and control application.

B. XMPP DEVELOPMENT

The Jabber Software Foundation's website, www.jabber.org, maintains a table that describes the development status of XMPP client and server projects. Table 8 describes the status of XMPP server development at the time of writing. As indicated by the table XMPP server implementations are available that are quite rich in features.

Server	Feature Score	License	Platforms
OpenIM	48%	BSD	AIX, HP-UX, Linux, MacOS X, Solaris, Windows
jabberd 2.x	76%	GPL	AIX, *BSD, HP-UX, Linux, MacOS X, Solaris, Windows
ejabberd	91%	GPL	AIX, *BSD, HP-UX, Linux, MacOS X, Solaris, Windows
psyced	59%	GPL	AIX, *BSD, HP-UX, Linux, MacOS X, Solaris, Windows
xmppd.py	21%	GPL	Linux
jabberd 1.x	45%	GPL	AIX, *BSD, HP-UX, Linux, MacOS X, Solaris, Windows
Wildfire	98%	GPL or Proprietary	AIX, HP-UX, Linux, MacOS X, Solaris, Windows
Merak	76%	Proprietary	Linux, Windows
SoapBox Server	78%	Proprietary	Windows
Antepo OPN	88%	Proprietary	AIX, HP-UX, Linux, Solaris, Windows
Jabber XCP	97%	Proprietary	Linux, Solaris, Windows
Sun Java System Instant Messaging	67%	Proprietary	HP-UX, Linux, Solaris, Windows
TIMP.NET	76%	Proprietary	Windows

Table 8. Status of XMPP Server implementations. (From: www.jabber.org/software/servers.shtml. Accessed 24 August 2006)

The XMPP server implementation used to support this research was the Wildfire server from Jive Software. Previously named Jive Server, Wildfire is an open-source XMPP server implementation written in Java. Wildfire was attractive for conducting this research for a number of reasons: the portability of Java™-written applications, the server's outstanding web-based administration tool, and, its rich set of features.

There are several dozen client applications written to support Chat/IM over XMPP. Clients have been written in most common programming languages, and there are available clients for nearly all computing platforms, including hand-held devices (Saint-Andre and Meijer, 2005). This research primarily used three client applications: Exodus, Spark, and the evolving JFCOM BuddySpace/Transverse, a project of Joint Forces Command aimed at building an XMPP client application with military-specific features built in.

There are also a number of programming-library implementations of XMPP. Application Programming Interfaces (API) for XMPP have been written in Java, Python, C, C++, .Net, Perl, Ruby, and Erlang. This research involved the use of the Smack open-source Java API for XMPP.

C. CLIENT IMPLEMENTATION AND DEPLOYMENT

1. Exodus

Exodus is an open-source XMPP chat client designed to run on Microsoft Windows. Its use in the conduct of this research is the result of personal preference of the author with respect to features and interface. Its presentation is for the purpose of demonstrating generic Chat/IM use with XMPP.

a. Installation

The Exodus client is available for download at: http://www.jabberstudio.org/projects/exodus/releases. Installation is wizard driven and easy to complete.

b. Login

Once installed, starting the Exodus Client opens a login window as seen in Figure 9. This is typical as all XMPP users must connect to a server.

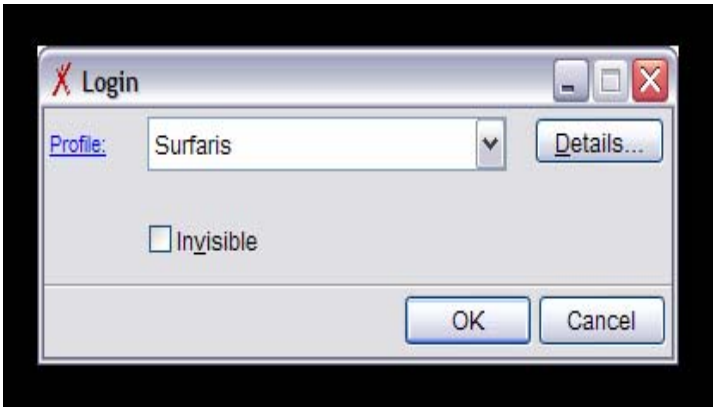

Figure 9. Starting the Exodus client initiates a login window.

Clicking the "Details" button permits configuration of login information as seen in Figure 10. This presents a window that allows user input for the JID, password, resource, and priority settings. Additional tabs allow for user setting of port number for the connection, identification of a proxy server, or to configure HTTP polling for a particular session. Exodus allows the creation of multiple account profiles, providing login support for multiple users and multiple accounts. This is a useful feature in military environments where multiple users regularly access computers.

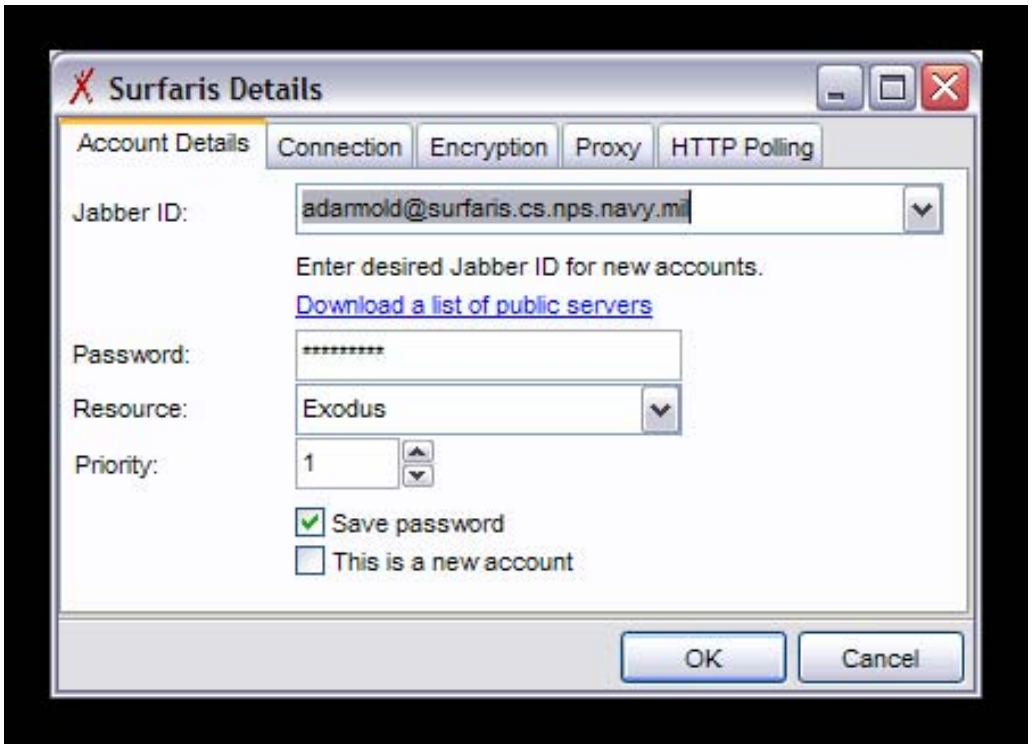

Figure 10. The Account Details tab of the Exodus login account details window allow for profile configuration.

c. Navigation and Use

Once logged in, the Exodus interface presents an interface that displays any stored offline messages as seen in Figure 11. The “Exodus” menu item allows for the sending of instant messages, starting a one-on-one chat or joining a chat room.

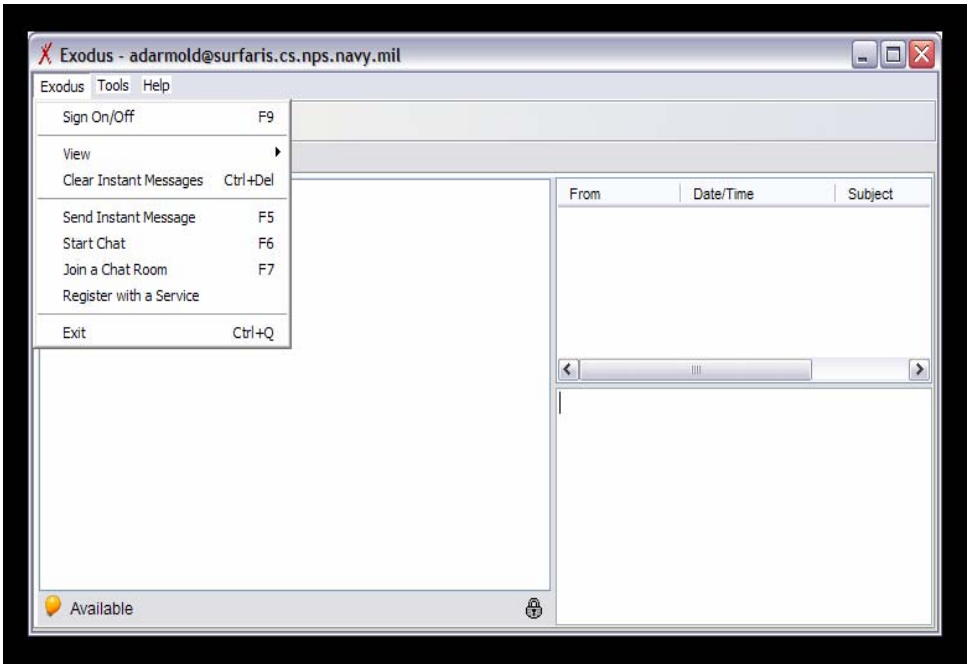

Figure 11. The initial Exodus Client interface is presented once the use has logged into the XMPP server.

In addition to the menu items, Exodus offers a browser feature that affords interface mechanics similar to a windows web browser. Figures 12 and 13 depict this feature.

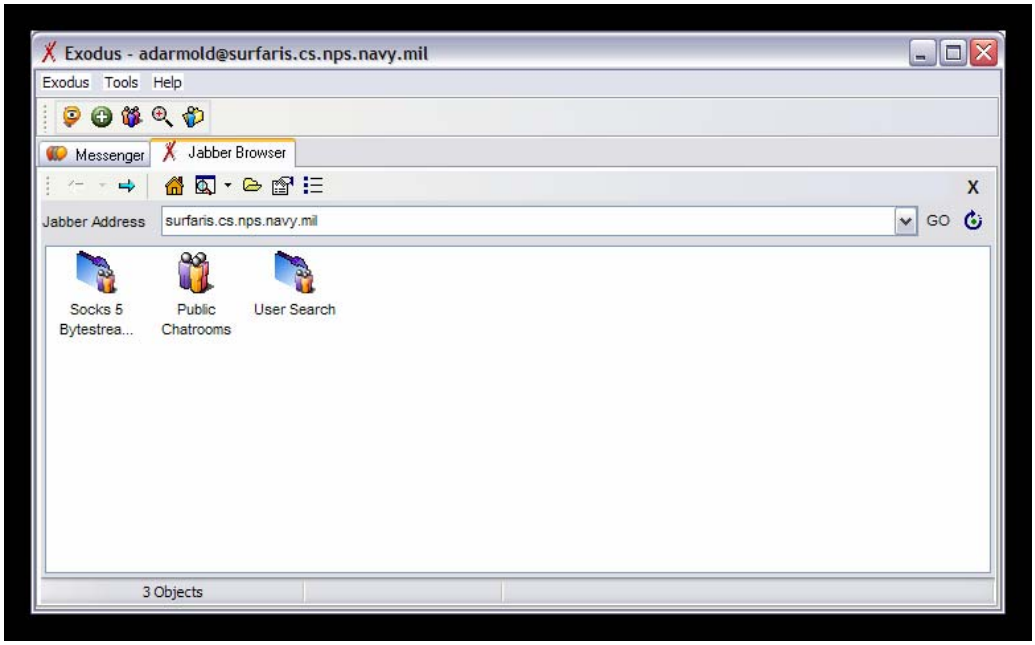

Figure 12. The Exodus browser interface allows for intuitive navigation to chat rooms or other users.

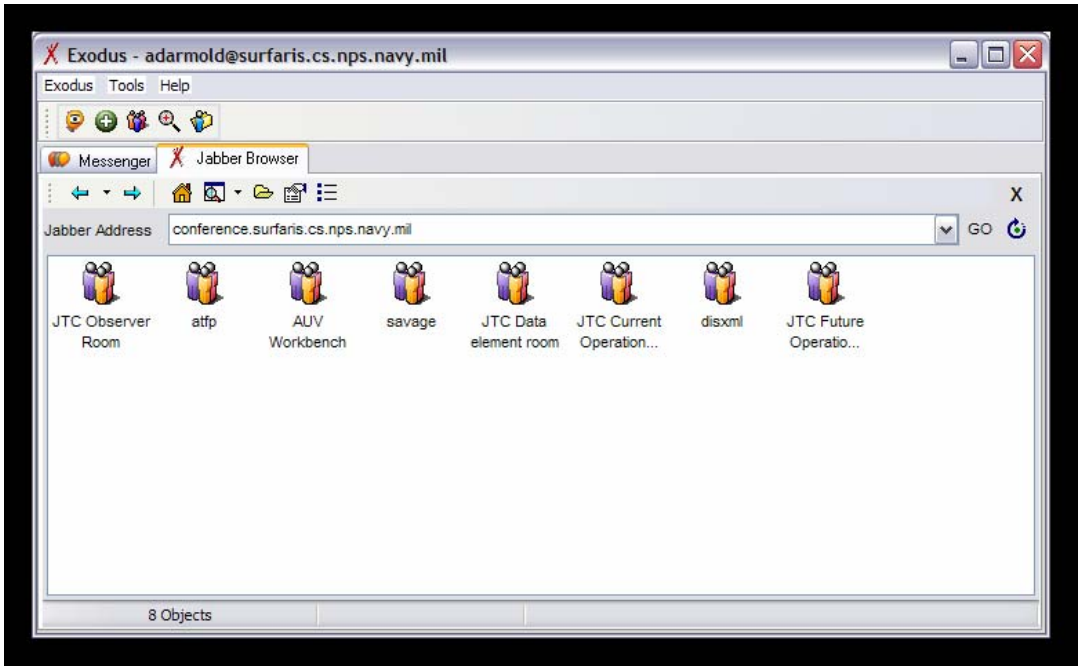

Figure 13. The Exodus browser feature provides intuitive display and navigation of the XMPP server's chat rooms.

All chat clients offer some form of contact list and roster group maintenance. These allow for presence awareness of some group of people that a user would frequently interact with. Roster groups allow for organization of the contact list

and message broadcast to a specified audience. In the simplest form, presence awareness will indicate who is online and logged into the XMPP network, but more advanced indicators of chatting activity (such as “do not disturb”) are available in most clients. Basic presence awareness is an important aspect of military chat and IM, but many of the advanced presence indicators might be categorized as convenience features rather than needs. Nevertheless, presence features may be applied in a meaningful way, such as indicating watchstander activity or monitoring-software health. Figure 14 displays some basic presence indicators of Exodus. Figures 15 and 16 display Exodus’ interface for one-on-one chat and a chat room.

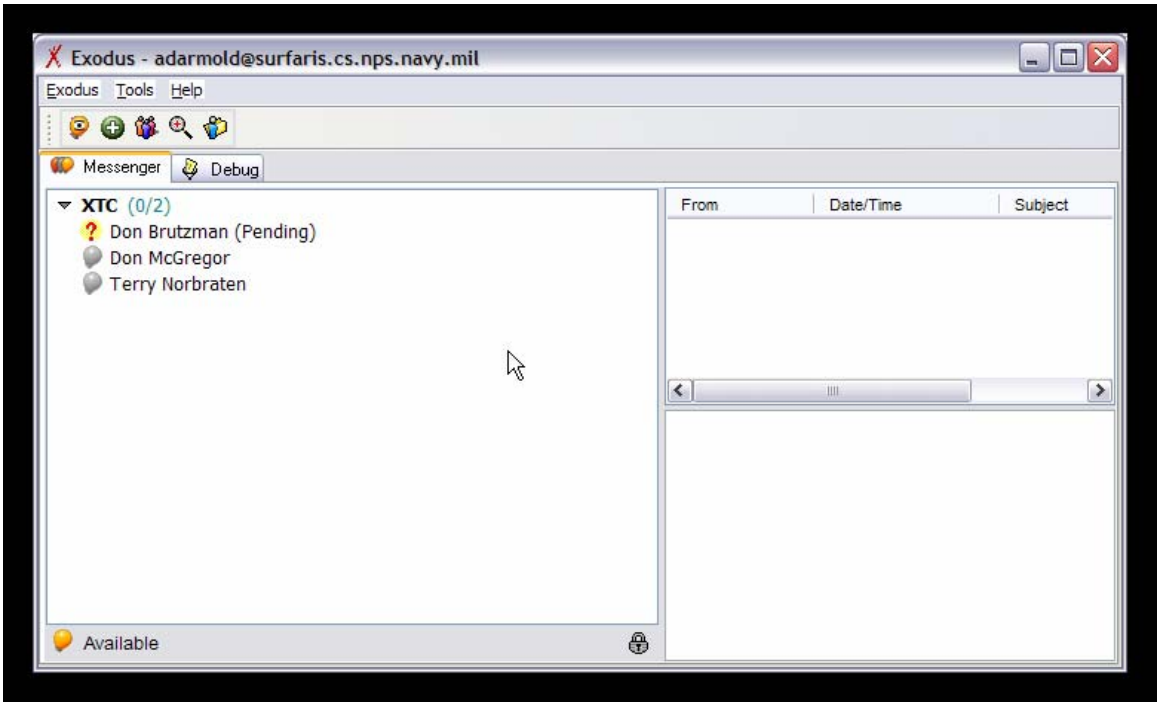

Figure 14. Exodus client displays presence indicators as seen with the contacts in the XTC Roster Group.

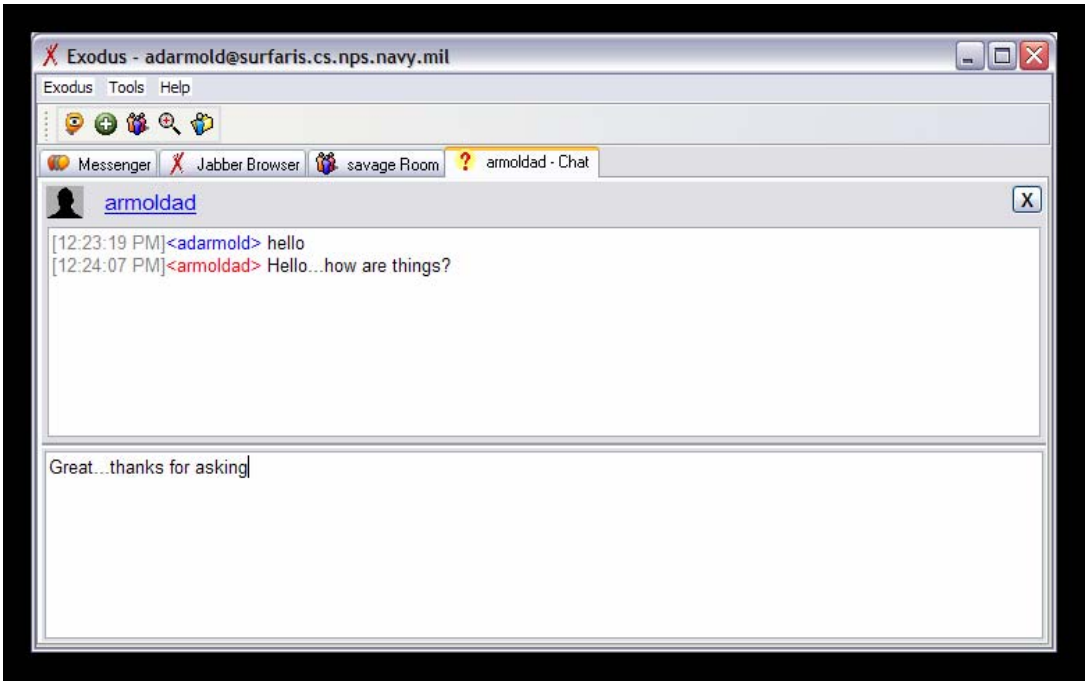

Figure 15. The Exodus client displays a two-frame window for one-on-one chat communications. The upper frame displays the conversation and the lower frame is for inputting text messages.

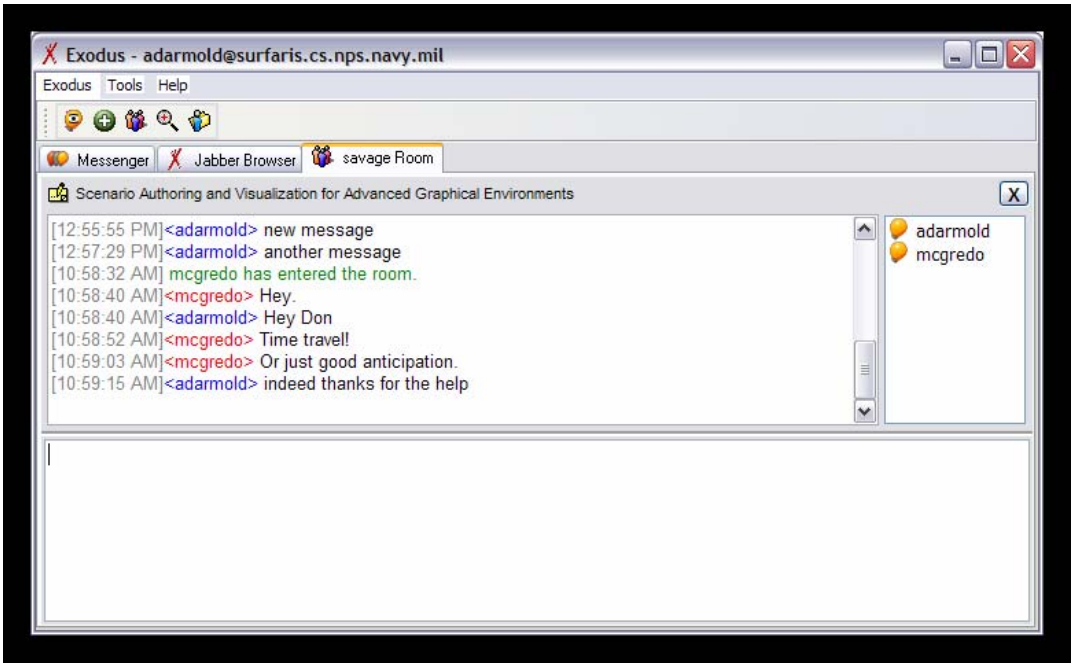

Figure 16. The Exodus client displays a two-frame window for multi-user chat communications. The upper frame displays the conversation and the lower frame is for inputting text messages.

Exodus is an easy to use, well-featured XMPP client. It affords multiple user login settings and provides a number of user friendly methods for discovering users and public chat rooms. However, there are many military specific communications needs with respect to messages traffic that, if applied to the Chat/IM domain, would greatly enhance Chat/IM value in the military context.

2. USJFCOM BuddySpace/Transverse

While commercially available Chat and Instant Messaging has proven itself to be a valuable method of communication for the military, a customized chat client for military purposes has great potential value. Much of the information passed in military messages requires security labeling, priority labeling, specific time-stamping, and other associated information items. As such, a customized XMPP Chat/IM client, named Transverse, has been built by the United States Joint Forces Command (JFCOM) as part of a larger collaboration system called the Cross Domain Collaborative Information Environment (CDCIE) Collaboration Tool.

a. Supported Chat Features

The JFCOM Transverse chat client originally was an extension of the BuddySpace open-source client. The project later re-built the tool using Jive Software's Smack open Java API for XMPP clients, though much of the look and feel of Transverse still comes from its Buddyspace roots. There are many of features in Transverse that support military application of chat communications. Some of these features are implementations of a basic chat client interface found to be useful in the military environment. These are listed below: (Fletcher, Lirrette, and Bishop, 2006)

- Buddy Lists
- Online users tab
- A MUC (group chat) room that contains all users using the client that are connected to a given server.
- Places tab
- A building->floor->room representation of rooms
- HyperRooms- A specialized room that contains other rooms. Allows users to monitor and participate in multiple chat rooms within a single window.
- Automated discovery of a server's available chat rooms

- Ability dock/undock windows
- User interface is themeable and supports language localization

Figures 17 and 18 display the contact or buddy list and the online users interface of Transverse.

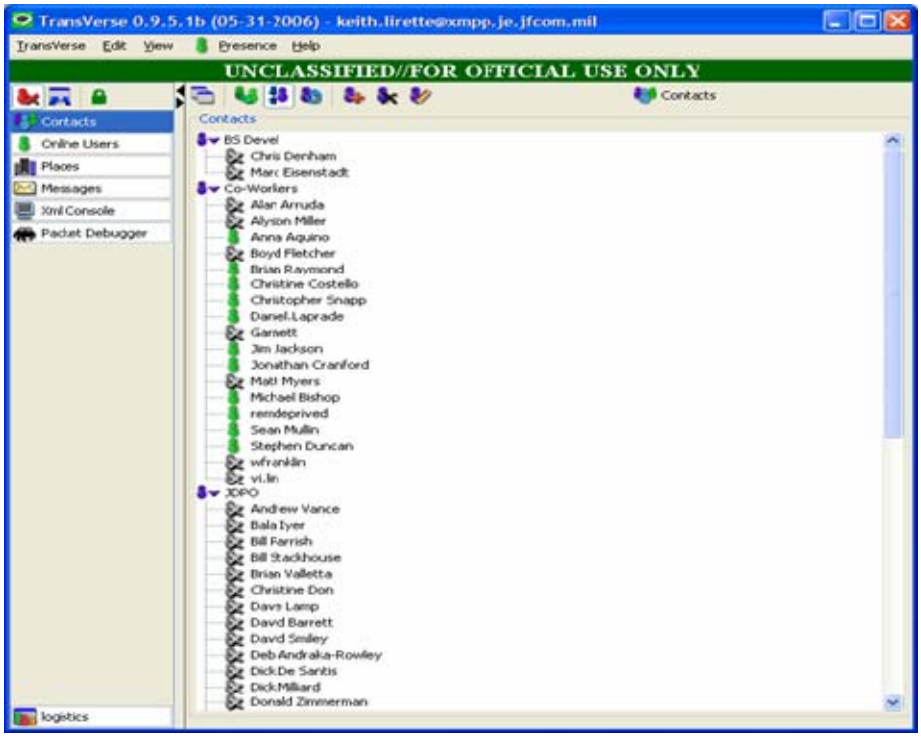

Figure 17. Transverse uses a Contacts or Buddy List display to support user awareness of other XMPP users. Classification labeling is for demonstration purposes only.

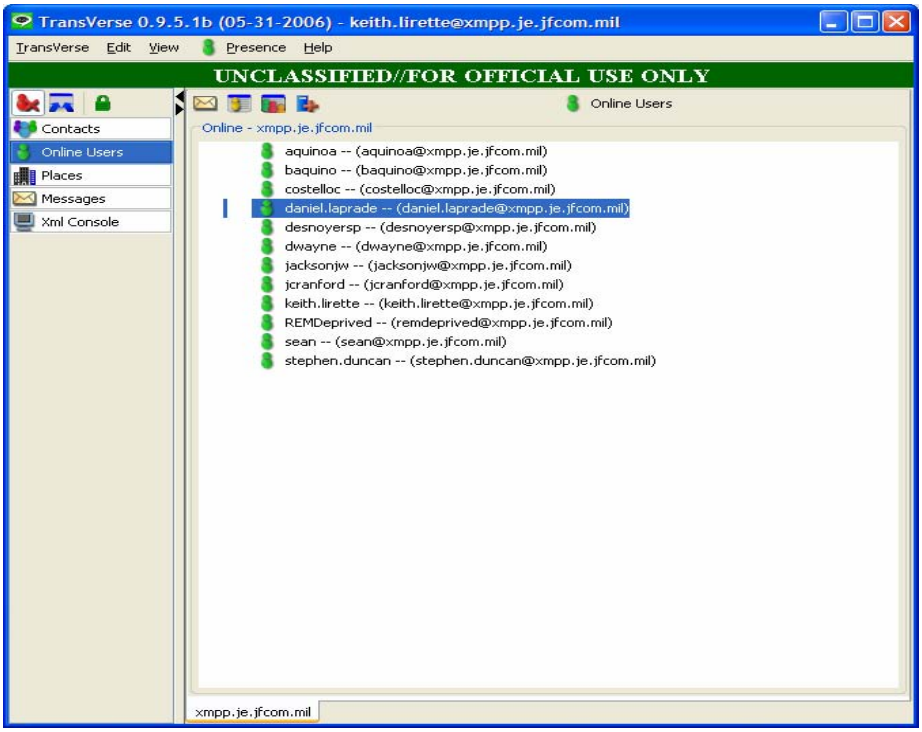

Figure 18. Transverse uses an Online User display to support user awareness of other connected XMPP users. Classification labeling is for demonstration purposes only.

Transverse, like most XMPP clients, allows the viewing of supported chat rooms on the connected server, as displayed in Figure 19. With Transverse, however, it is also possible to arrange these chat rooms into logical hierarchical groupings. This allows the chat rooms to be arranged by the same organizational structure as the functions the rooms are supporting, a useful feature in the military environment. This is seen in Figure 20.

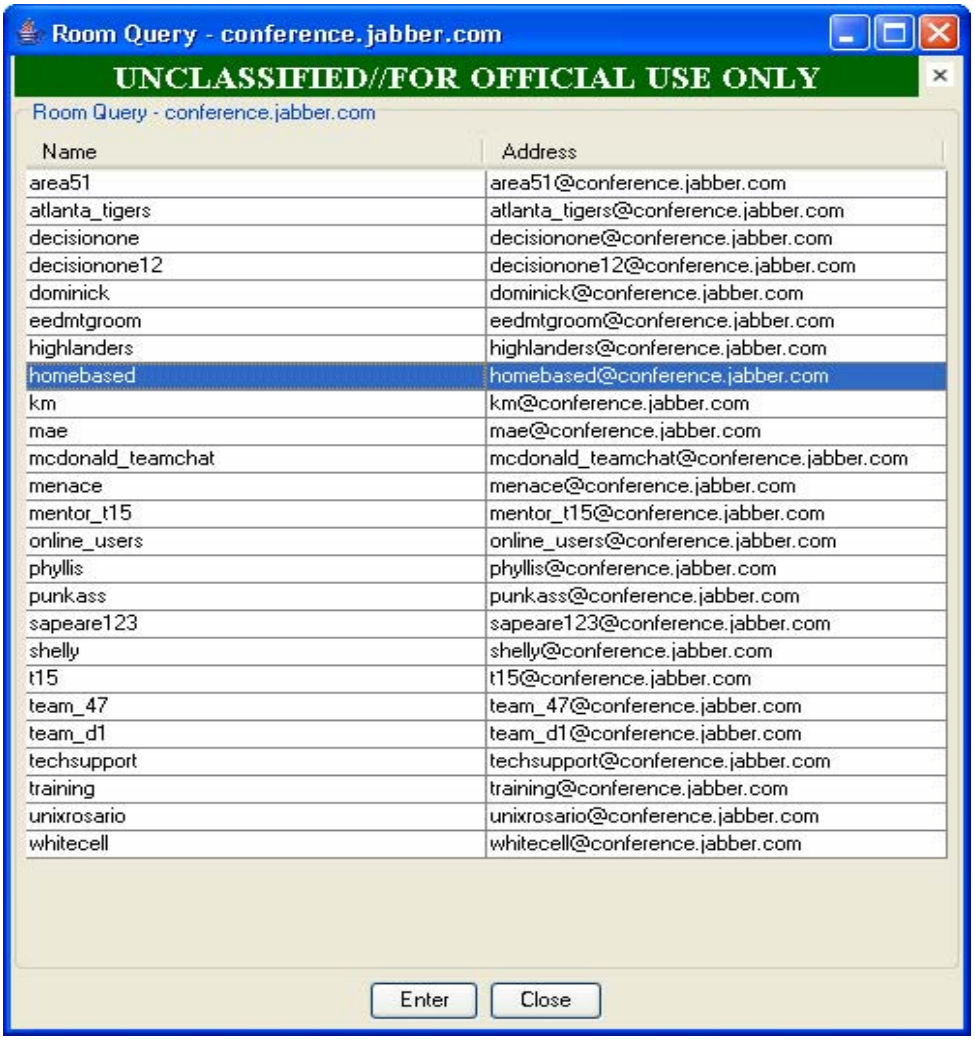

Figure 19. Transverse supports the display of a Room Query on the connected server. This figure displays the room list for conference.jabber.com. Classification labeling is for demonstration purposes only.

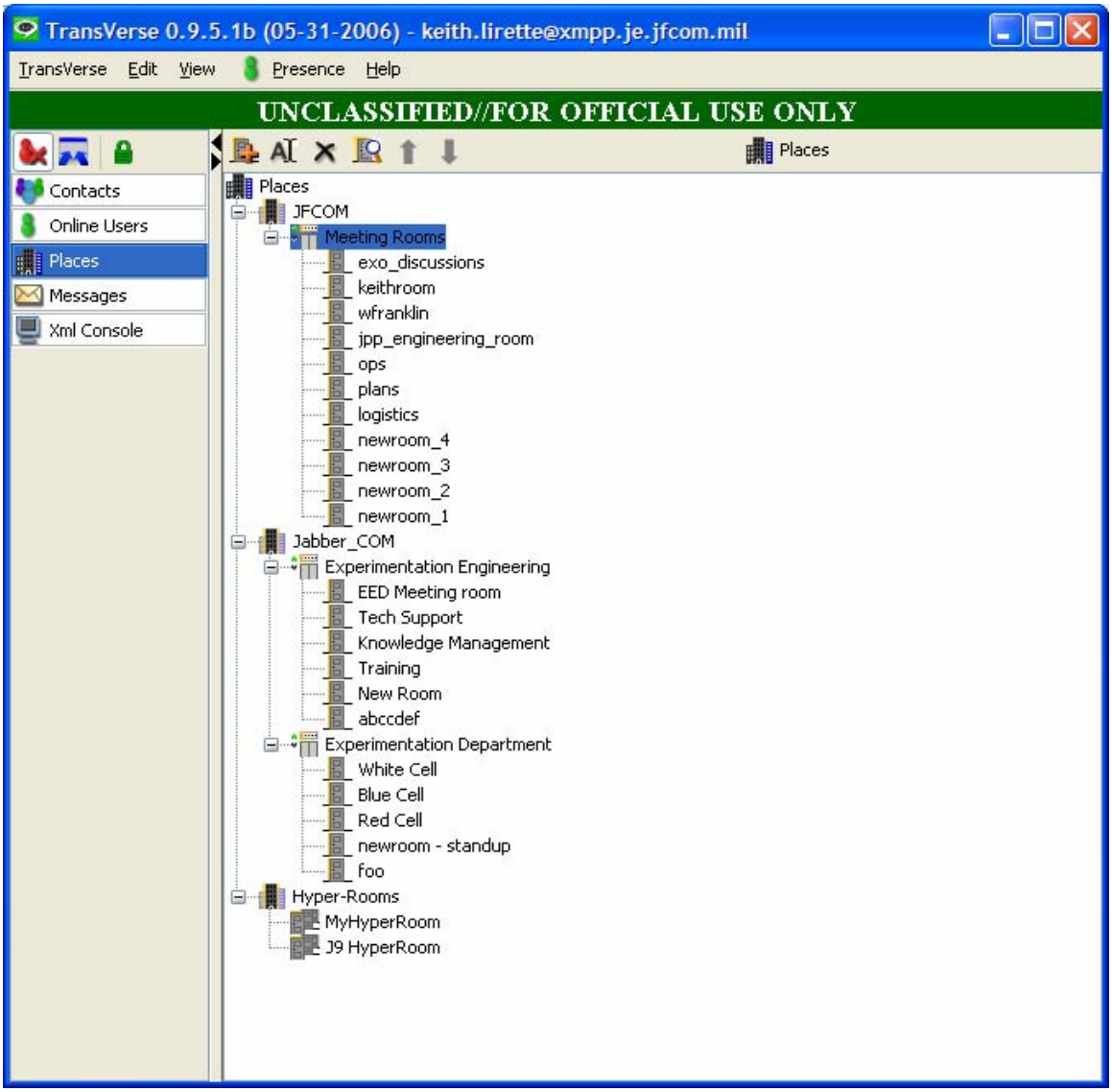

Figure 20. Transverse supports the logical organization of chat rooms into groups. Classification labeling is for demonstration purposes only.

b. Supported Military Specific Features

In addition to modified implementations of basic chat client features, Transverse has other features that are specifically aimed at supporting military chat communications in a multi-national environment. These are listed below: (Fletcher et al., 2006)

- Concurrent keyword monitoring of multiple chat rooms
- Classification labeling of chat messages. The labeling of chat messages when used with an associated Collaboration Gateway will let users participate in cross classification domain group text chat sessions.

- HyperRoom Implementation allow for the consolidation of multiple chat room monitoring in a single GUI window
- Enhanced Chat Message Window that displays username, time stamp, classification, original/pivot/destination languages, digital signature and encryption notification, and room name (if in HyperRoom mode) in boxed display
- Language Translation using CyberTrans II
    - Supports automatic translation of inbound and outbound messages without user intervention
    - Supports manual translation of inbound and outbound messages
    - Supports language pivoting - using one language to translate between two other languages for which a direct translation does not exist.
- Full logging of all chat sessions on client. Most XMPP servers also have the ability to log chat sessions
- Object (message) level digital signing using W3 XML Digital Signature Specifications

Figure 21 is a display of Transverse's message keyword monitoring capability. This feature allows a user to select one or more chat rooms to monitor and list of one or more keywords to listen for. If one of the keywords appears in a monitored room, the user is alerted. Many military users are required to multi-task functions on their computers. This feature allows the user to focus on other tasks, while maintaining alertness to the chat rooms with respect to any topics of interest.

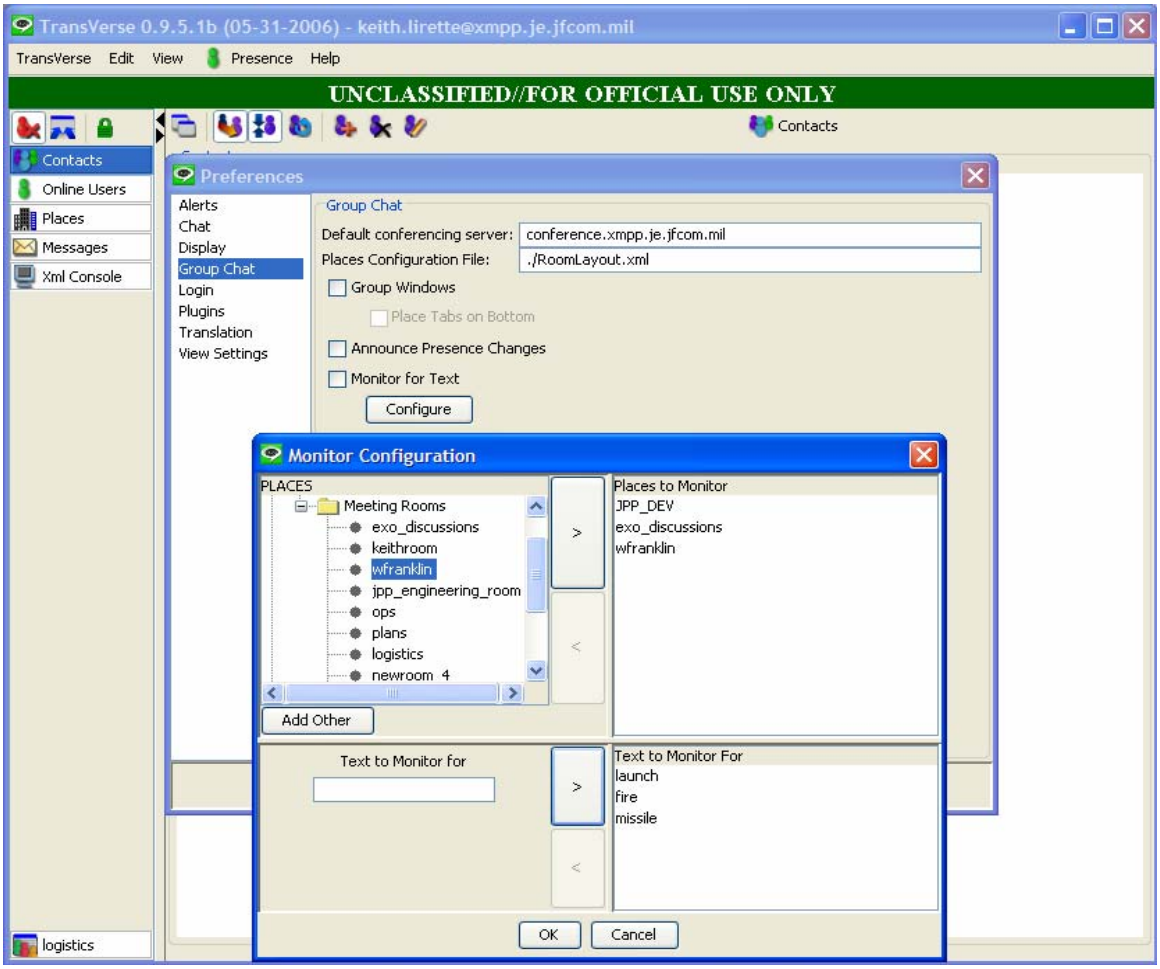

Figure 21. Message Keyword Monitoring is supported by Transverse. Note the ability to select multiple rooms as well as multiple keywords for alerting. Classification labeling is for demonstration purposes only.

Military messages, whether passed over a computer network or across radio systems, are typically categorized with respect to classification. It is a goal of the DoD to move toward collapsing the existing separate networks for classified, multinational, and unclassified information systems. Any communications system that would integrate into such network must have the ability to label such categorizations. The Transverse client provides this ability as depicted in Figure 22.

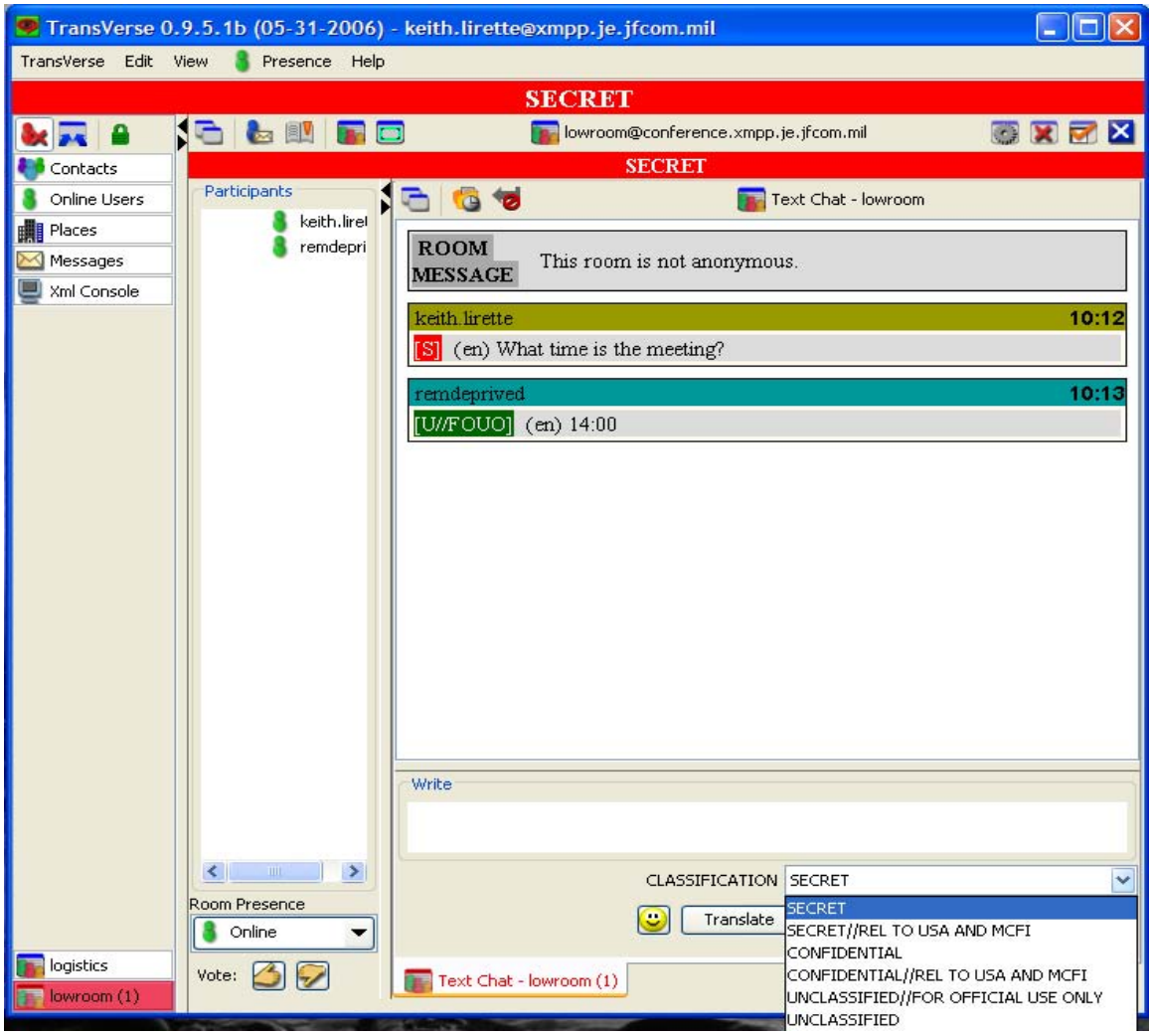

Figure 22. Transverse Chat client enables labeling and display of message classification information. Classification labeling is for demonstration purposes only.

Transverse also has a feature called HyperRooms. HyperRooms allow a user to monitor, send messages to, and receive messages from multiple chat rooms while working from a single GUI window. This feature, like the message monitoring feature, allows for more efficient use of the computer interface for Chat. There are many cases in the military where HyperRooms would be useful. A unit's Fire Support Coordinator (FSC), for example, is required to coordinate and de-conflict the conduct of fire originating from multiple platforms (mortars, artillery, aviation). These platforms may employ chat room communications, and the FSC may desire awareness of the chat activity in each of these rooms. The HyperRoom feature allows for the consolidation of

chat windows into a single frame, without losing any awareness or contextual information. Furthermore, it frees up monitor space for other tasks. Figure 23 displays Transverse's HyperRoom feature.

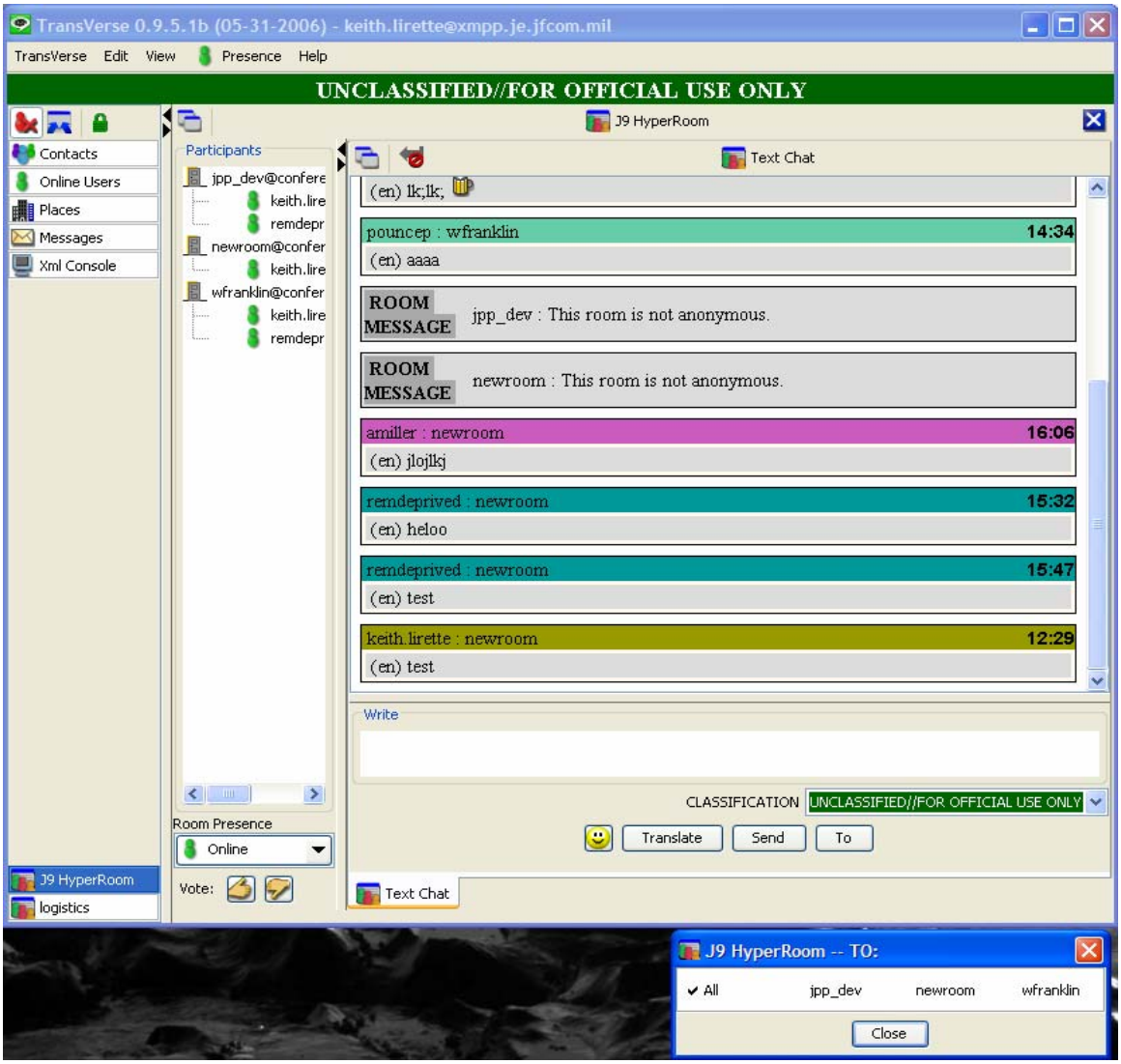

Figure 23. Transverse's HyperRoom feature. This window is supporting three different chat rooms, wfranklin, jpp_dev, and newroom. Note the ability to select which room/s the user can send a particular message to. Classification labeling is for demonstration purposes only.

The ability log, store, and retrieve communication information is a key requirement to all military Information Technology (IT) systems. The standard for implementing server-side chat logging is still in the experimental status as a Jabber

Enhancement Proposal. Client side message logging is an important feature for any military chat client. Transverse enables client side message logging and retrieval as depicted in Figure 24.

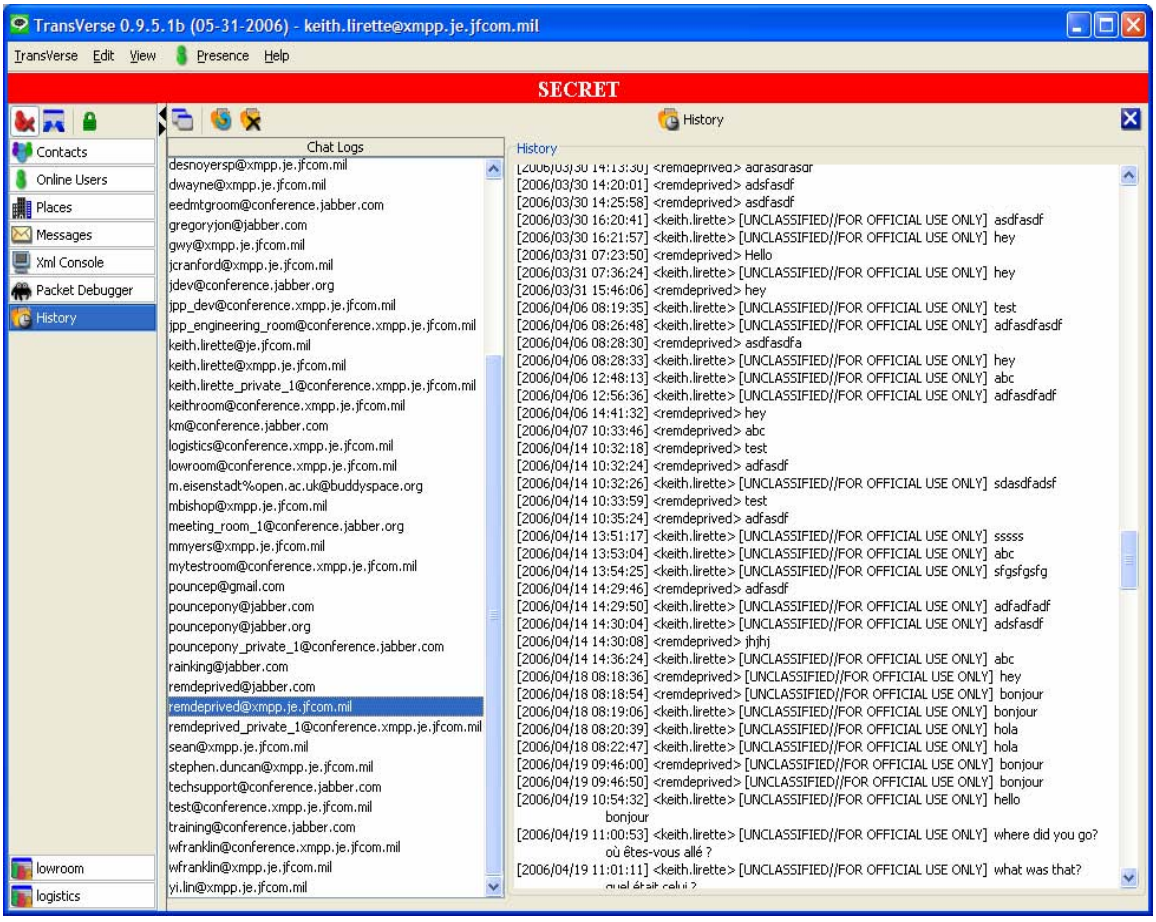

Figure 24. Transverse enables both one-on-one and chat room logging and retrieval. Classification labeling is for demonstration purposes only.

One of the biggest barriers to multinational coalition operations is that of language. Many of the United States' allies are not native English speakers. Command and Control, along with all other warfighting functions, are adversely affected by language barriers. Computer supported language translation tools aim to provide a solution to this problem. Transverse interfaces with a language translation tool, and permits both the manual and automatic translation of chat messages. Figures 25 and 26 depict manual language translation with the Transverse tool, and Figure 27 displays a capture of the automatic language translation.

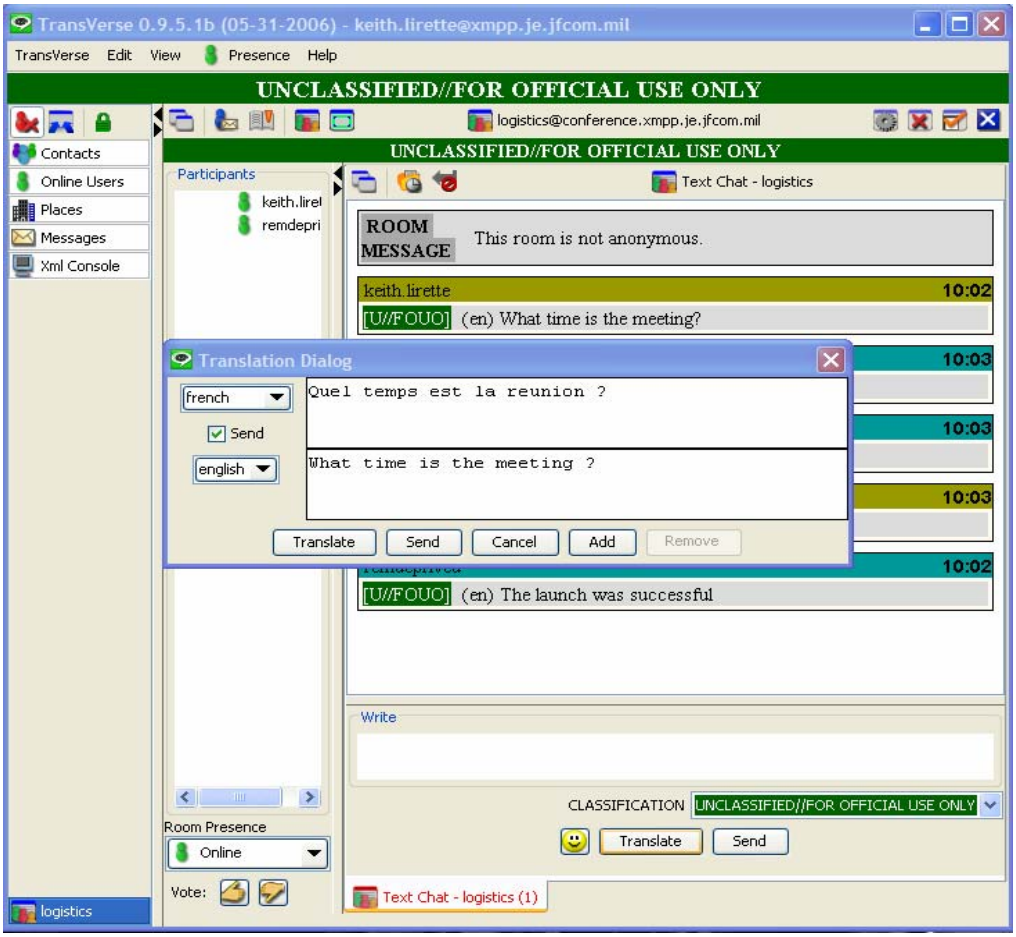

Figure 25. Transverse's manual language translation tool allows a user to selectively translate individual messages. Classification labeling is for demonstration purposes only.

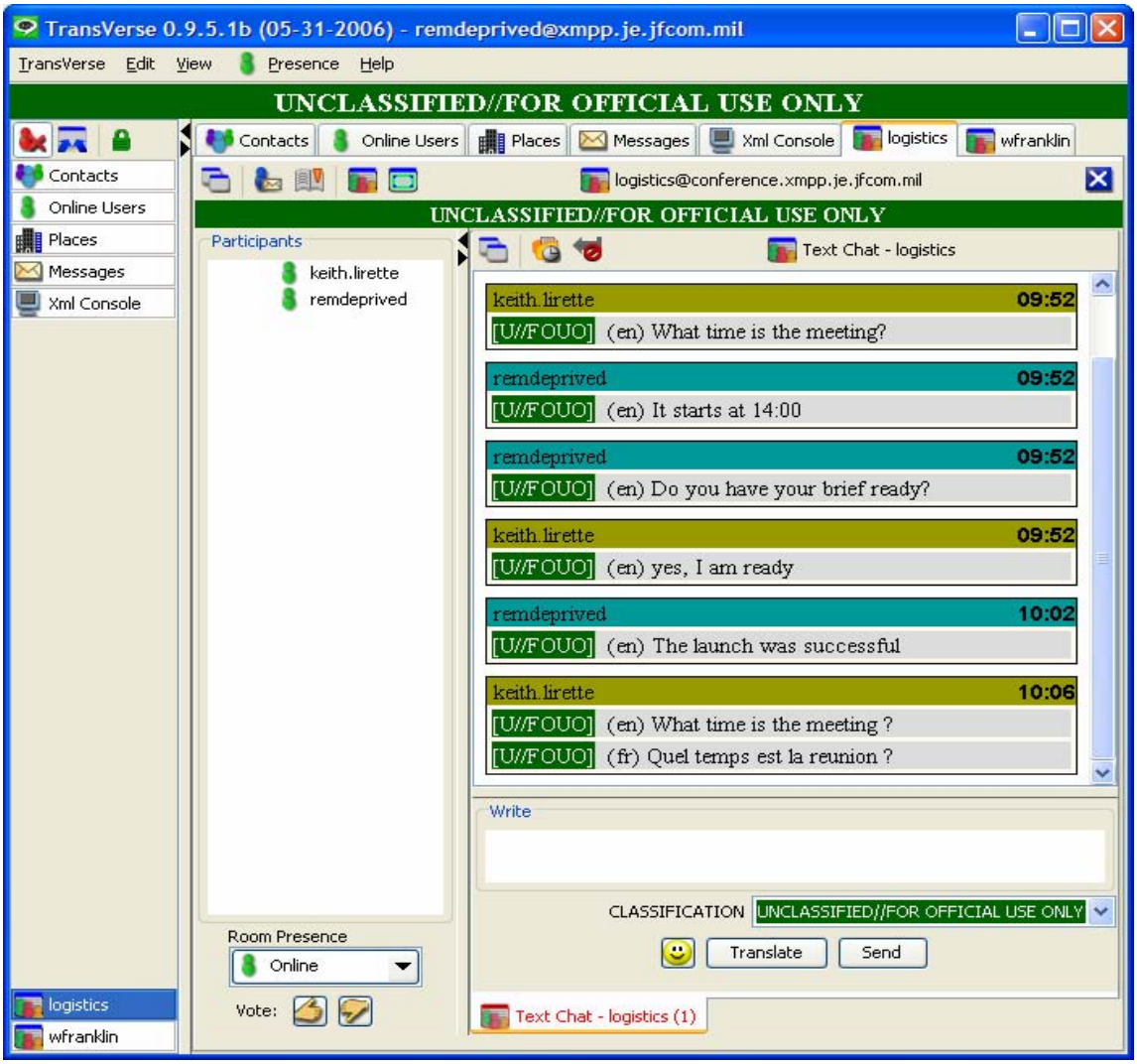

Figure 26. A manually translated chat message displayed in a Transverse Chat Room. Classification labeling is for demonstration purposes only.

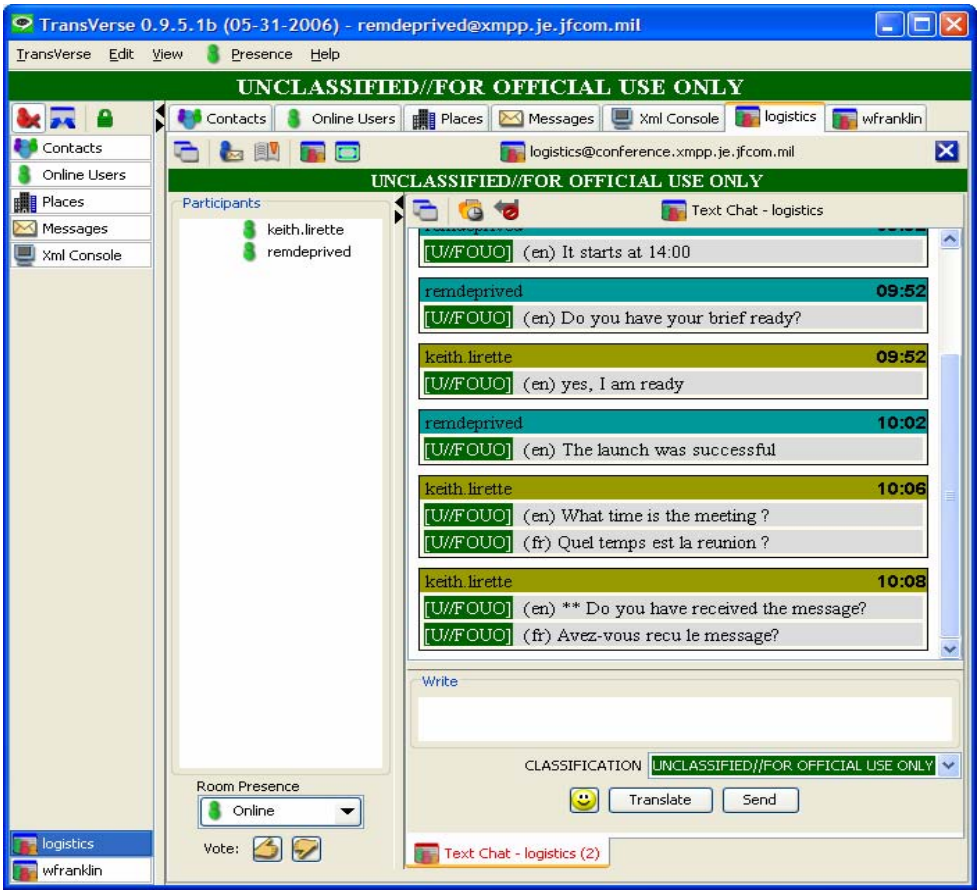

Figure 27. Transverse Chat room display showing automatic language translation. The translated message is denoted with **. Classification labeling is for demonstration purposes only.

3. Military Chat Customization

The use of customized chat clients will likely continue in the DoD. JFCOM's Transverse is an excellent example of such a tool, enabling many features that military users will find useful and efficient, as well as integrating an interface for adding military message label to chat and IM communications. Transverse stands as a strong example of the inherent potential in adopting XMPP standards as the basis for a chat communications network. Future military-purposed chat tools should be developed to support organizational needs in the areas of:

- Enhanced Language Translation
- Whiteboarding
- Audio and Application Casting (one-way sharing) support
- Integration into Command and Control suites.

D. XMPP SERVER IMPLEMENTATION AND DEPLOYMENT

As noted earlier, Wildfire Server has a web-based administration tool that makes configuration, maintenance, troubleshooting and management of the XMPP server and users quite easy. This section presents some of the important and useful features of the Wildfire Server.

1. Wildfire Server Settings and Features

The Wildfire Server Admin Console allows configuration of the server settings. This is useful for system troubleshooting and maintenance of the server. Figure 28 displays the server settings page of the server manager. Other functions with respect to server settings administration are system properties settings, plug-in support, file transfer settings, compression settings, private data storage configuration, resource conflict handling, and registration and login settings. It is also possible to enable or disable anonymous login, password changing, and in-band account registration, which allows a new user to create an account from the client side.

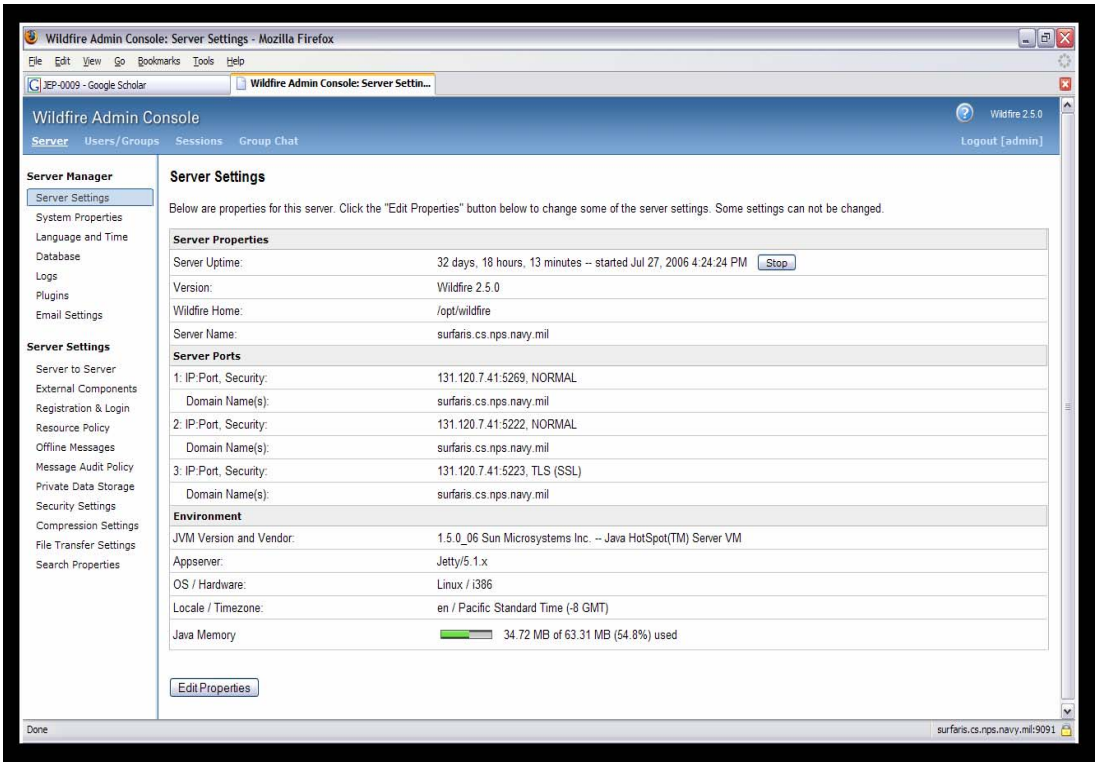

Figure 28. The Wildfire Admin Console Server Settings Page displays configured settings and server status.

One of the more useful admin console features is the Log Viewer. Through the console an administrator is able view error, warning and, information logs. Additionally, there is a debug log viewer that can be enable or disabled through the console. The debug logs display the XML traffic handled by the server and are useful for troubleshooting XMPP connections and traffic, as seen in Figure 29.

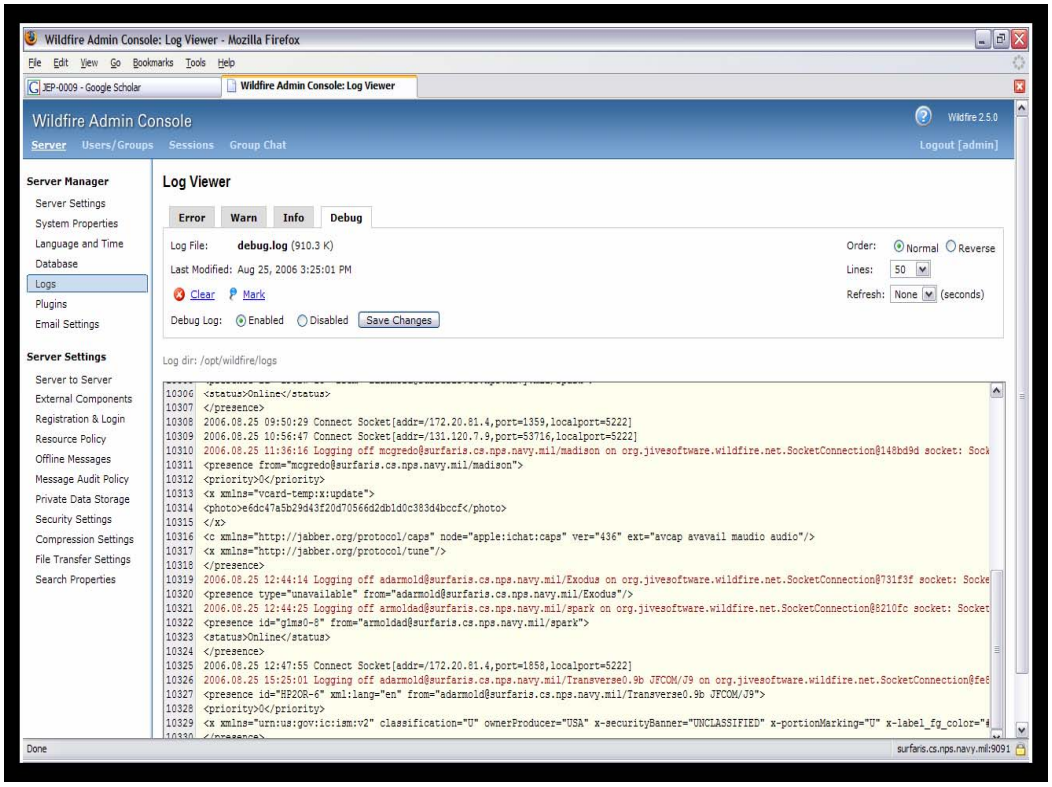

Figure 29. The Wildfire Admin Console Debug Log Viewer provides log access to the XMPP traffic and facilitates troubleshooting and debugging.

Another useful tool, both for troubleshooting and for storage and retrieval of XMPP information, is the Message Auditing capability of Wildfire. Figure 30 depicts the console window which permits setting the message audit policy for the server. If message auditing is selected, the admin user can select the type of stanzas to collect, as well as configure file settings for the logged XML files.

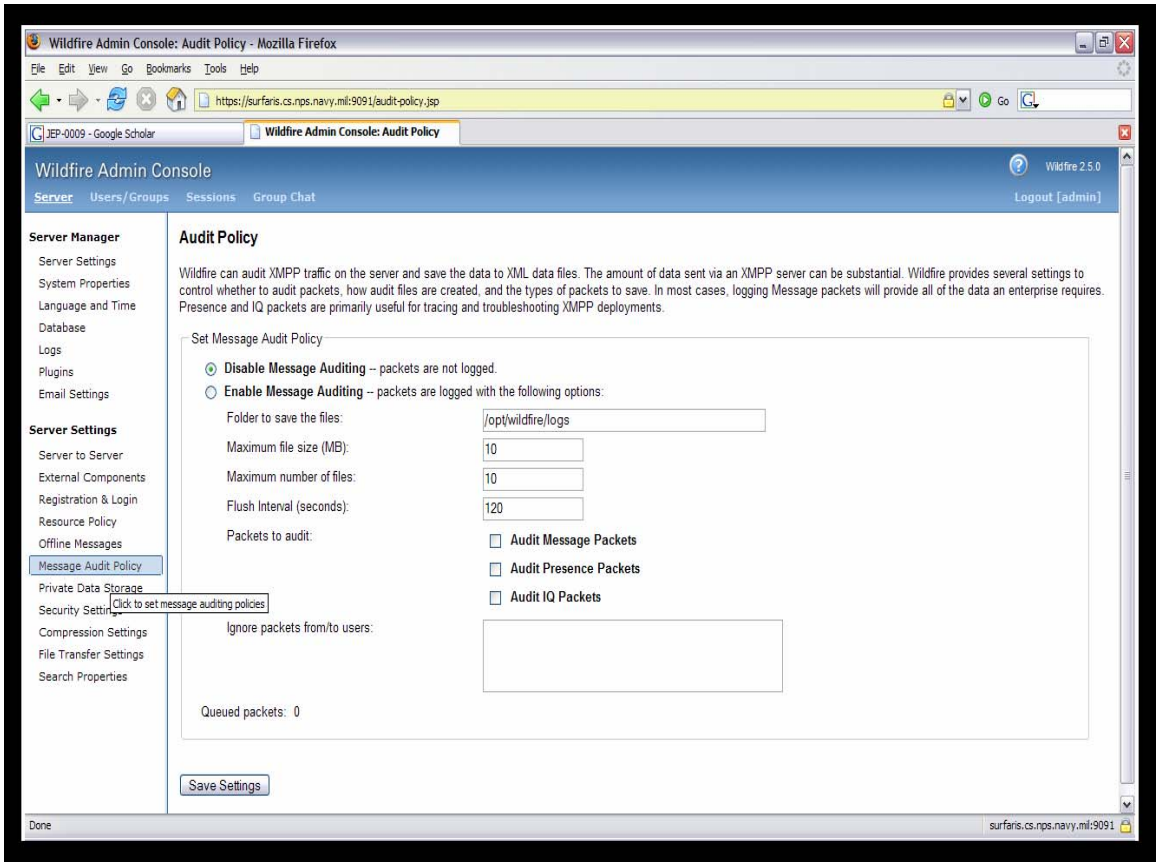

Figure 30. Wildfire Admin Console: Message Auditing Policy provides the ability to enable or disable packet logging and auditing. Note the ability to select the type of packet (stanza) types to audit and log.

Wildfire has an off-line message policy configuration tool that allows for great flexibility in the handling of messages sent to users who are not connected to the XMPP network. With this console feature, it is possible to configure the XMPP server to store these messages until the user connects to the server, then deliver the messages. This is a necessary feature when seeking to implement XMPP in an unstable network environment as is seen in wireless tactical networks. The off-line message feature also allows for notification to the sender of recipient unavailability if off-line storage is disabled. A recommended additional feature that would be useful in the military context is to enable off-line message storage and delivery, but also implement sender notification of the initial unavailability of the receiver and subsequent delivery of the message. This can ensure delivery of the message and maximize the sender's knowledge of recipient awareness. Figure 31 is the admin console's display for off-line message handling.

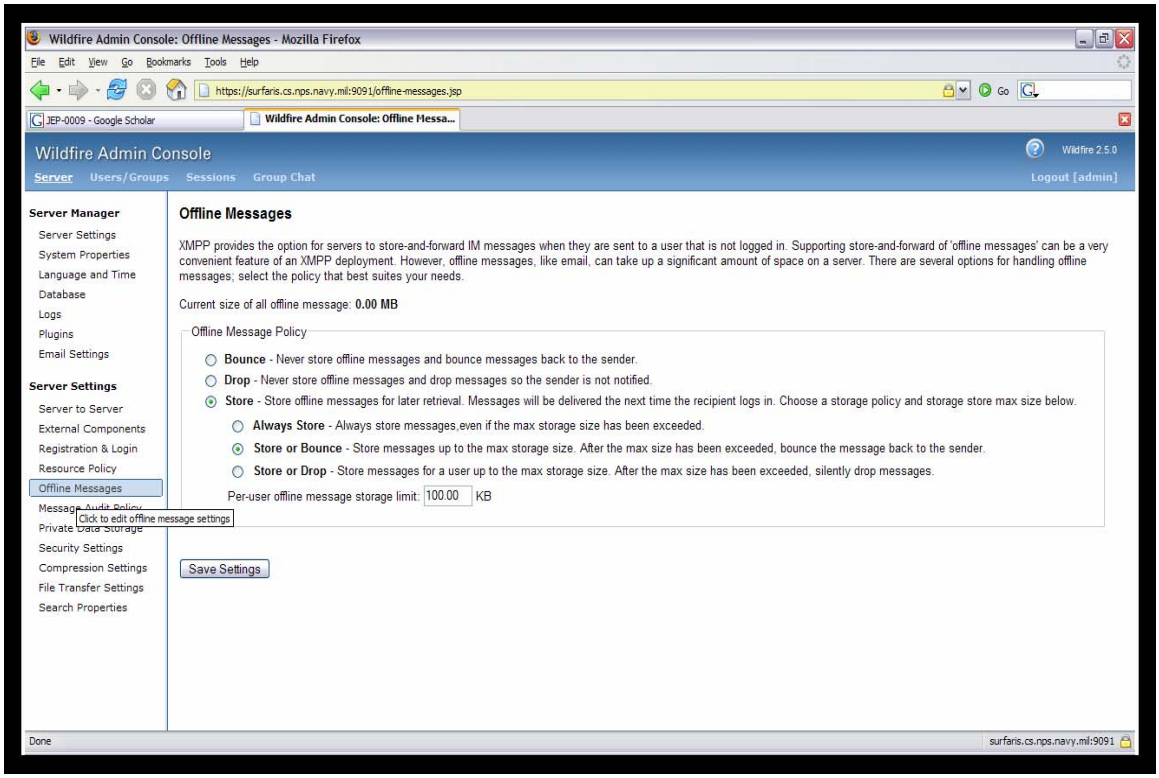

Figure 31. The Wildfire Admin Console: Offline Messages page enables configuration of off-line message handling.

2. User Accounts and Multi-User Chat Rooms

User account management is also enabled by the Wildfire Admin Console. From the console, an administrator can create, delete, set passwords for, and modify the properties of user accounts. User groups can be established and configured from the console as well. Figure 32 displays the console's user account management display.

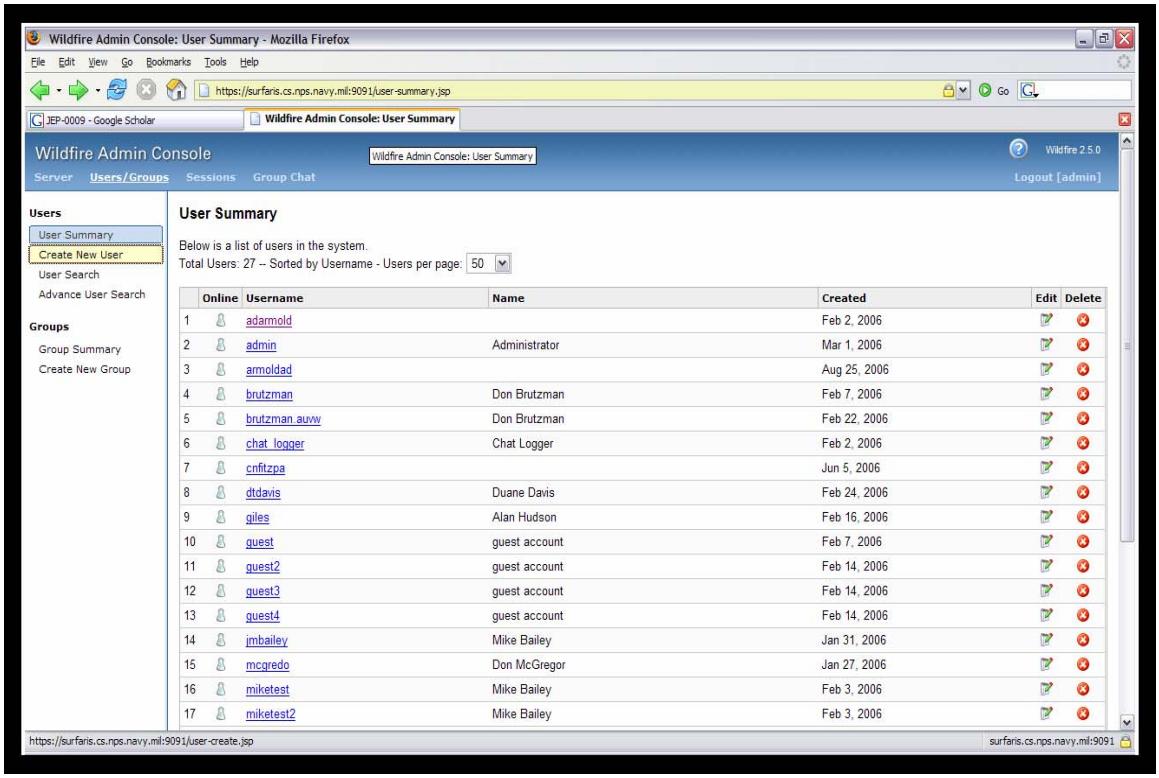

Figure 32. Through the Wildfire Admin Console: User Summary, administrators can create, delete, and modify user accounts with this feature.

The Wildfire Console allows for the configuration and management of the Multi-User Chat rooms on the Wildfire server. This is shown in Figures 33 and 34. This feature allows administrators to control room access, set room history display settings, create room administrators, and configure settings for logging chat room conversations and kicking idle users in the chat rooms. The Wildfire console allows for the creation and configuration of MUC rooms with regard to persistence, rules for nickname use, password protection, and other features.

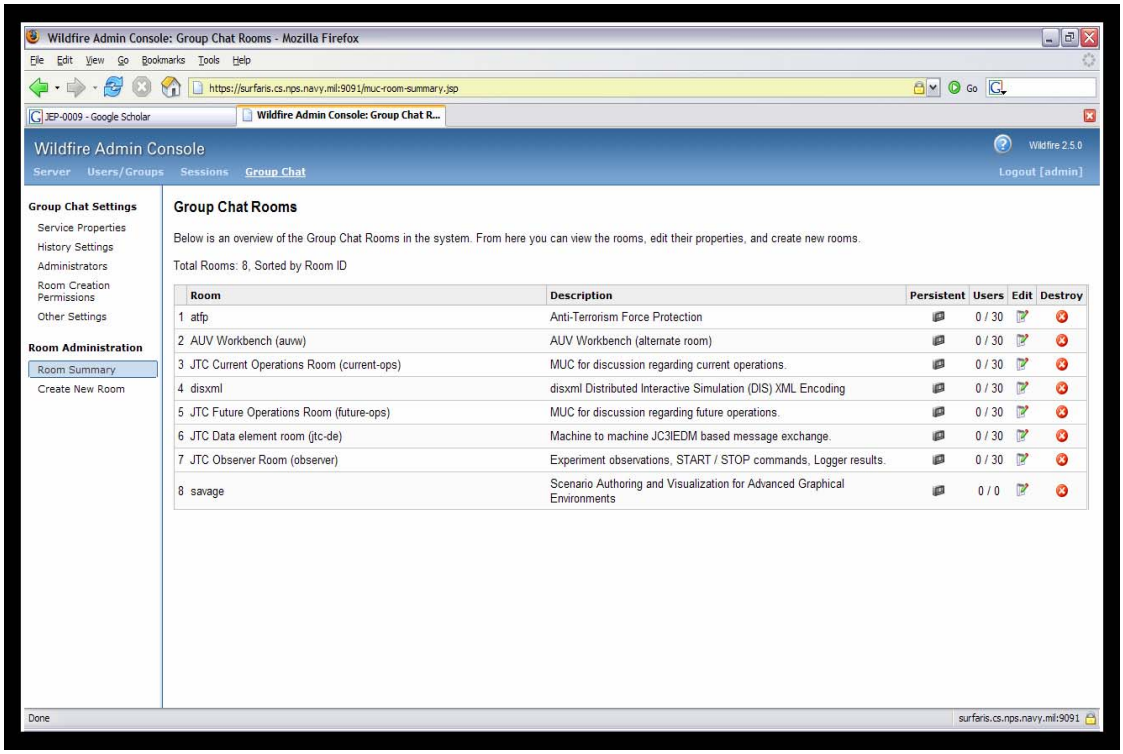

Figure 33. The Wildfire Admin Console: Group Chat Rooms page displays the status of all supported MUCs on the server.

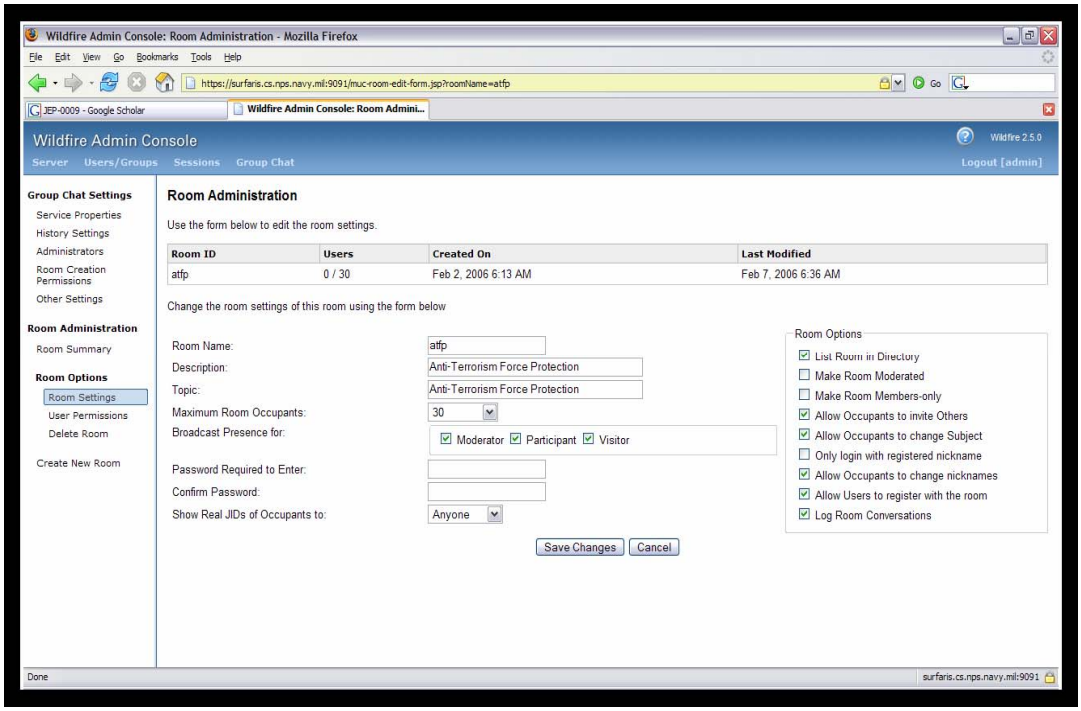

Figure 34. The Wildfire Admin Console: Room Administration page allows for configuration of MUC rooms. Note the available room options for MUC room configuration control.

3. Server Connections

Finally, the Wildfire Console provides administrator visibility on the XMPP connections that the server is supporting. Figures 35 and 36 depict session display for client sessions and server sessions respectively.

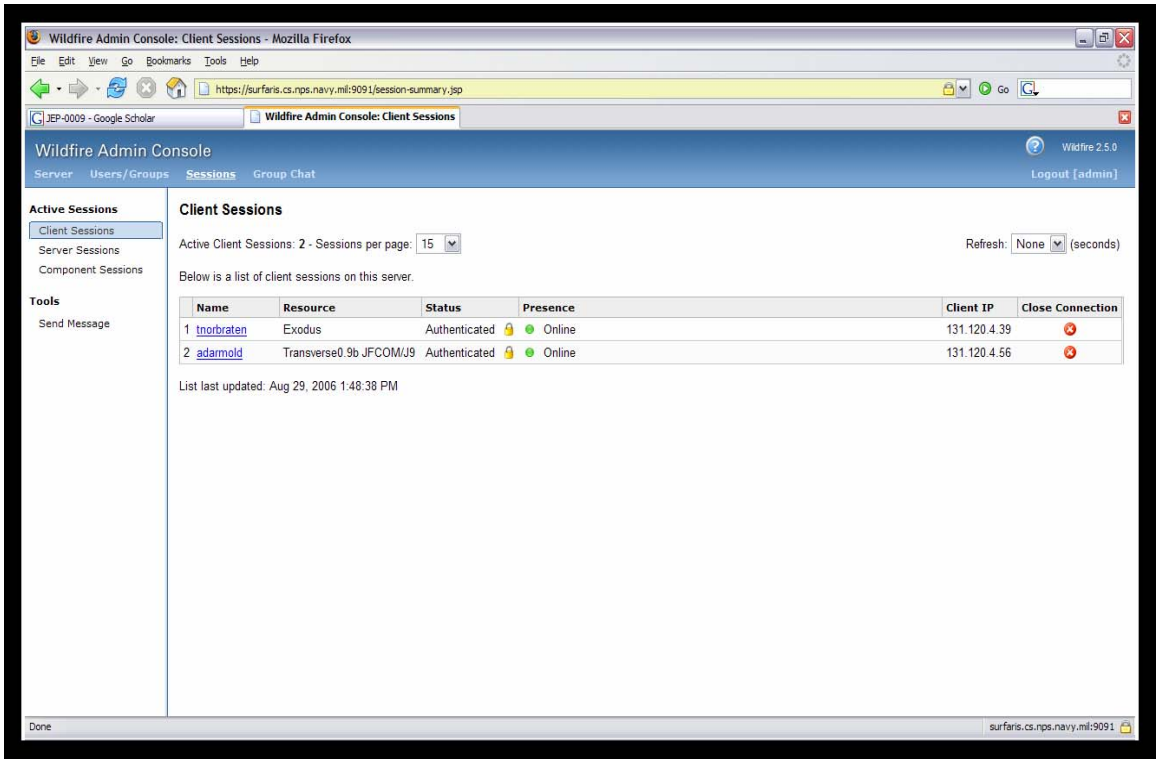

Figure 35. The Wildfire Admin Console: Client Sessions page displays the status of all connected clients.

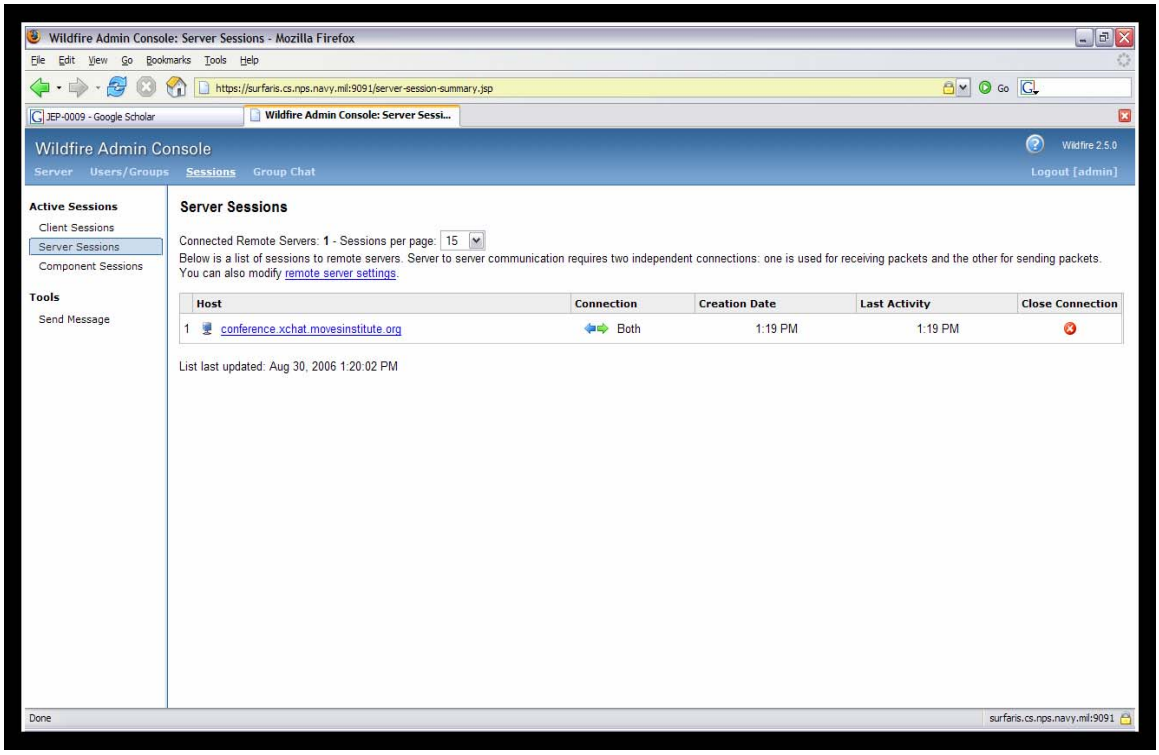

Figure 36. The Wildfire Admin Console: Server Sessions page displays the status of all active server to server connections.

4. Military Considerations

This research found the open-source Wildfire XMPP server to be a stable platform with many desirable features for configuration management. The web-based administration console was an extremely useful feature in the conduct of the XTC research. Many military organizations may not have the information management personnel to implement an open-source XMPP server, but there are several well-supported commercial XMPP server products that offer the same feature set as the Wildfire server. Important to these implementations is the ability to easily administer server settings. Proper military application of Chat and IM will require active management of user accounts and MUC rooms. A tool, such as the Wildfire Admin Console, will prove quite valuable for the administration of the military chat network.

E. SMACK XMPP CLIENT LIBRARY

The final tool used in the course of this research was the Smack Open-Source Java API for XMPP clients. Though capable of implementing full XMPP clients, this research used Smack to allow other Java processes to connect to an XMPP network to

perform some function. The author found Smack to be concise, easy to understand, simple to apply, well-documented, and complete enough to meet the research needs of the project. Some of the library's mechanisms are discussed further.

1. Connection and Login

Smack provides a simple method for connecting to an XMPP server. Figure 37 displays some examples for establishing an XMPP connection with Smack (Jive Software, 2006). Once instantiated, the XMPPConnection object has a method that enables logging onto the server.

```
// Create a connection to the jabber.org server.
XMPPConnection conn1 = new XMPPConnection("jabber.org");

// Create a connection to the jabber.org server on a specific port.
XMPPConnection conn2 = new XMPPConnection("jabber.org", 5222);

// Create an SSL connection to jabber.org.
XMPPConnection connection = new SSLXMPPConnection("jabber.org");
```

Figure 37. Establishing an XMPP connection with Smack is done by instantiating an XMPPConnection class object.

2. Messaging

The Smack API allows programmers to send and receiver XMPP messages. Figure 38 describes how to connect to an XMPP server and send a message with only a few lines of Java code. Alternatively, it is possible to build XMPP messages that contain specialized content, or to create MUC messages by instantiating a GroupChat class object (Jive Software, 2006).

```
// Assume we've created an XMPPConnection name "connection".
Chat newChat = connection.createChat("jsmith@jivesoftware.com");
newChat.sendMessage("Howdy!");
```

Figure 38. Establishing a Chat session and sending a message with Smack can be performed with very few lines of code.

3. Stanza Processing

Smack groups the XMPP stanzas as a Packet class. The library has extensive capability for packet handling, enabling an application to listen for, filter, and process

incoming traffic as required. There are two constructs that provide this ability: org.jivesoftware.smack.PacketListener (PacketListener), an interface that will listen for incoming packets and process them in some fashion, enables event style programming, and org.jivesoftware.smack.PacketCollector, a class that lets you wait for a specific packet to arrive (Jive Software, 2006). This research used the PacketListener interface extensively.

Smack also contains a number of packet filters that provide more precise packet handling procedures. The default set of filters the library provides is listed below: (Jive Software, 2006)

- `PacketTypeFilter` -- filters for packets that are a particular Class type.
- `PacketIDFilter` -- filters for packets with a particular packet ID.
- `ThreadFilter` -- filters for message packets with a particular thread ID.
- `ToContainsFilter` -- filters for packets that are sent to a particular address.
- `FromContainsFilter` -- filters for packets that are sent to a particular address.
- `PacketExtensionFilter` -- filters for packets that have a particular packet extension.
- `AndFilter` -- implements the logical AND operation over two filters.
- `OrFilter` -- implements the logical OR operation over two filters.
- `NotFilter` -- implements the logical NOT operation on a filter.

4. Packet Extensions

Smack's packet extensions provide the mechanism for customized XML parsing of namespaced content inside the XMPP stanzas. This feature implements the extensibility feature of XMPP by enabling the network to be used as a generic XML router rather than just a Chat/IM path. The research presented in Chapter VII does not utilize packet extensions, but, in retrospect, should have been. Instead, machine-purposed XML was either attached as a packet property in the form of a Java string variable, or passed as the body of a MUC chat message. Properly implemented extensions of XMPP using Smack should register an extension provider and handle its XML payload independently of any Chat/IM parsing methods (Jive Software, 2006).

5. Packet Properties

In addition to providing for custom XML content handling, Smack also allow for the attachment of arbitrary properties to packets. These properties are either Java primitives or serializable Java objects. Some of the research presented in Chapter VII uses packet properties to attach XML data, as a Java String class object, to message packets. This proved to be an effective method because all nodes in the experiment were running Java. This methodology is not particularly scaleable and is not recommended for future implementations. Smack's use of packet properties is not standardized and will not easily interface with other XMPP clients, but does provide a simple mechanism for attaching arbitrary data to XMPP packets in controlled environments (Jive Software, 2006).

6. Smack Extensions

The Smack library packages the XMPP Core and IM/Presence functionality separately from the functions captured by the Jabber Enhancement Proposals. Table 9 lists the current extensions supported by Smack (Jive Software, 2006).

Name	JEP #	Description
Private Data	JEP-49	Manages private data.
XHTML Messages	JEP-71	Allows send and receiving formatted messages using XHTML.
Message Events	JEP-22	Requests and responds to message events.
Data Forms	JEP-4	Allows to gather data using Forms.
Multi User Chat	JEP-45	Allows configuration of, participation in, and administration of individual text-based conference rooms.
Roster Item Exchange	JEP-93	Allows roster data to be shared between users.
Time Exchange	JEP-90	Allows local time information to be shared between users.
Group Chat Invitations	N/A	Send invitations to other users to join a group chat room.
Service Discovery	JEP-30	Allows to discover services in XMPP entities.
File Transfer	JEP-96	Transfer files between two users over XMPP.

Table 9. Smack supports a number of XMPP Extensions.

F. SUMMARY

Since its inception in 1999 and its standardization in 2004, XMPP has matured as a technology. The standard has been widely adopted as a Chat/IM solution both commercially and in the open-source development community. In this chapter, a set of tools was exposed to demonstrate the some of the capabilities of available XMPP clients, servers, and development libraries. In their current state, XMPP implementations are capable of meeting the majority of military Chat/IM requirements. The open standard platform avails the possibility of building military specific tools, such as Transverse,

without fear of losing interoperability or functionality. As XMPP sees further adoption, it is expected that the standardized feature set will expand, providing capabilities that extend well beyond conventional chat and IM.

THIS PAGE INTENTIONALLY LEFT BLANK

VI. XML TACTICAL CHAT (XTC)

A. INTRODUCTION

XML Tactical Chat (XTC) is a research concept to explore the potential application of XMPP technology in the military. XTC has an infrastructure supported by the Modeling, Virtual Environments, and Simulation (MOVES) Institute at the Naval Postgraduate School (NPS). XTC research includes both internal experimentation and integrating research with other military organizations and operational exercises. This chapter discusses the architecture and major research areas of the XTC project.

B. NPS XTC CHAT CONFIGURATION

Though much XMPP use is conducted over open access servers to support individual chat and IM, implementation in the military will require a more restricted and structured XMPP network design. XTC is supported by a small XMPP architecture of two servers. As can be seen in Figure 39, one of these servers is physically located on the NPS campus and inside the school's firewall, while the other is physically located outside the campus, supported by a firewall of its own. This configuration allows for testing of permission requirements between organizational firewalls as might be required for military deployment.

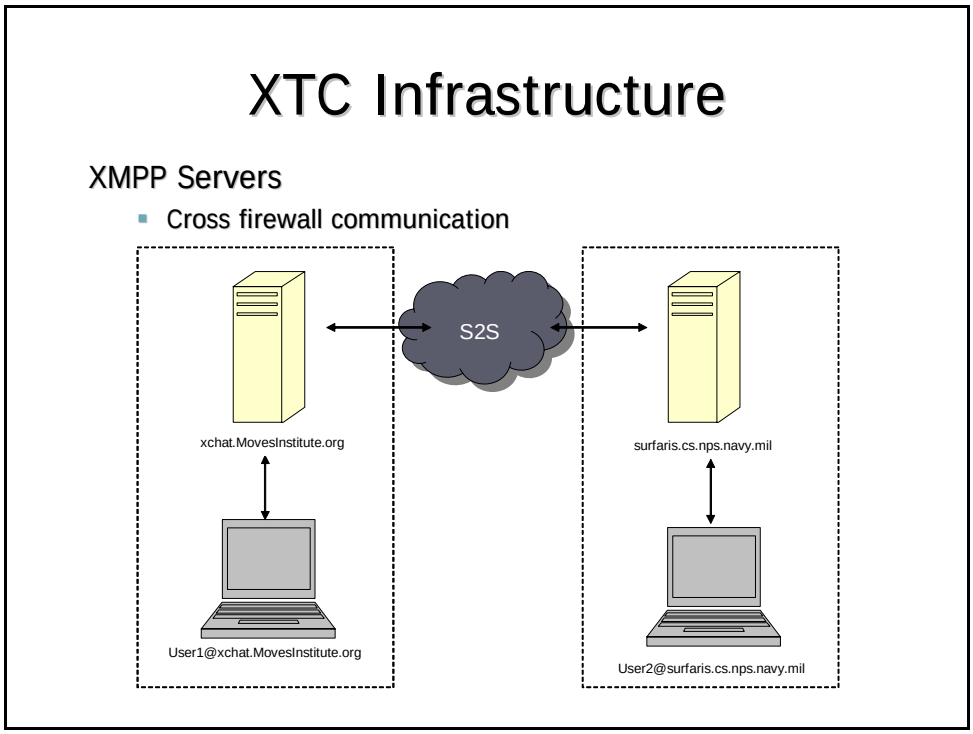

Figure 39. The NPS XTC configuration for chat enables XMPP communications across the campus firewall.

The on-campus server has the registered domain name of `surfaris.cs.nps.navy.mil`. The off-campus server is registered as `xchat.movesinstitute.org`. Table 10 lists the server specifications for these machines.

Server Name	Hardware	Operating System
surfaris.cs.nps.navy.mil	Dual 750 MHz Pentium 3 CPU 1 GB Internal Memory	Fedora Core 3 Linux
xchat.movesinstitute.org	Single 1GHz Pentium 3 CPU 512 MB Internal Memory	CentOS Linux

Table 10. XTC research is conducted on two servers with the listed specifications.

The XTC architecture has allowed for experimentation of XMPP chat across a number of network configurations. Provided the appropriate firewall permissions are in place, XMPP chat communications have been validated across and between the NPS LAN, wireless LAN, and through Cisco VPN portals.

Table 11 lists the NPS firewall permission configuration rules supporting XTC. Firewall permissions are set to allow any XMPP client user within the campus domain to access the specific IP addresses of XMPP servers outside the domain. Similarly, server to client traffic is permitted from specified XMPP server IP addresses outside the domain to any address within the NPS network. Server to server traffic is more tightly constrained, as traffic on port 5269 is permitted from specified XMPP server IP addresses inside the domain to specified XMPP server IP addresses outside the domain.

The XTC architecture has integrated with several other military XMPP networks in support of exercises and experiments. It is often necessary to temporarily establish both client-server and server-server connections between the two networks to assist in troubleshooting such efforts. In deployed environments, usually only server-server connections are required.

Port Numbers / Traffic	**Direction of Traffic**	**Permissions:**
5222/5223 / Server-Client	Inbound	Granted from IP to Network
5222/5223 / Client-Server	Outbound	Granted from Network to IP
5269 / Server-Server	Inbound	Granted from IP to IP
5269 / Server-Server	Outbound	Granted from IP to IP

Table 11. XTC traffic is permitted by the NPS firewall permissions.

The XMPP server software supporting the XTC project has varied. XTC has used, or attempted to use, the OpenIM Server, jabberd, Jive Server, and Wildfire. Installation of the OpenIM server was not successful. XTC was supported for a period of several months with the jabberd server. During this time, the server required frequent maintenance and re-booting. Installation and maintenance of the Jive Server (later

renamed Wildfire) resulted in greater stability and ease of use. At the time of writing, the surfaris.cs.nps.navy.mil XMPP server is Jive Software's open-source Wildfire version 2.5.0 and the xchat.movesinstitute.org XMPP server is Wildfire version 3.0.1.

NPS has recently restructured internal campus networking to establish the nps.edu domain in addition to the nps.navy.mil domain. Campus users on nps.edu have access rights equivalent to other navy.mil hosts through the application of routing configurations, internal firewall policies and reverse DNS lookup. Future work for NPS XTC includes examination of how to best extend the NPS XMPP server to server configuration employed in this thesis across the .mil extranet, .mil/.edu intranets, and the .edu extranet.

C. DIS-XML

As discussed in Chapter II, Distributed Interactive Simulation is an Institute of Electrical and Electronic Engineers (IEEE) binary standard protocol for conducting simulation across multiple platforms. One of the research areas of XTC is the conversion of binary DIS to an XML equivalent and the use of XMPP to transport DIS-XML across and between networks. Exposing DIS as XML would be advantageous by availing the XML data processing toolset to DIS data; storage, query, transformation, web services, and other XML processing capabilities become possible.

IEEE binary DIS data is passed on a wire in the form of discrete units called Protocol Data Units (PDUs). Traditionally, DIS has been implemented by creating a programming language object binding to the binary DIS PDU. Rather than attempt to create a direct binding between IEEE DIS and XML, this research seeks to use IEEE DIS binding to Java classes as an intermediate step for conversion to XML. An XML schema for IEEE DIS PDUs and a Java/XML binding protocol are added to complete the conversion. Figure 40 diagrams this process.

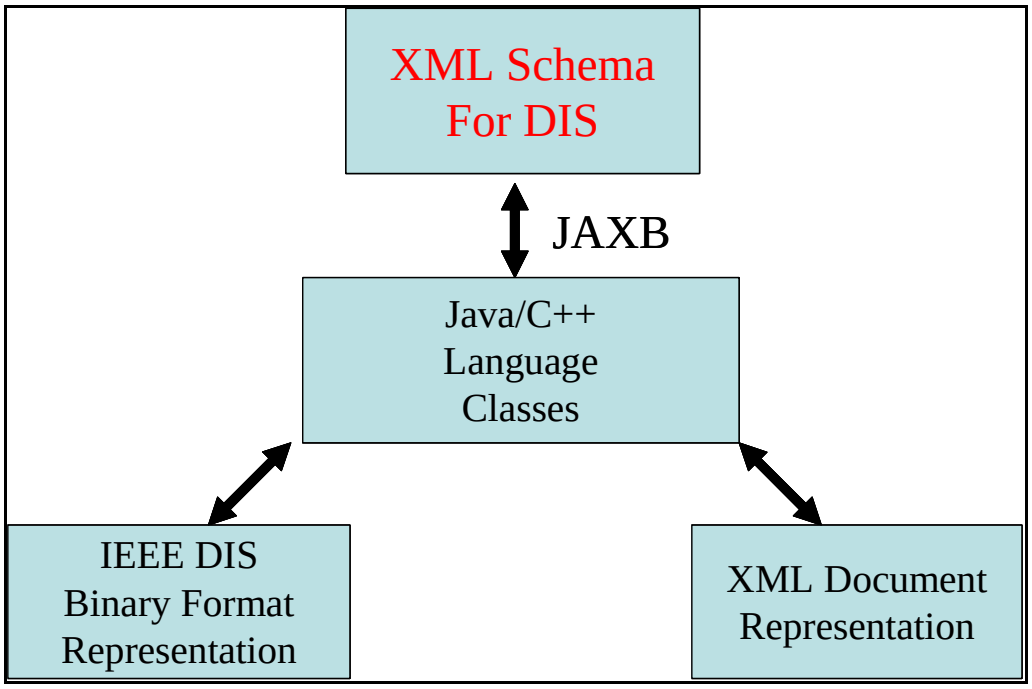

Figure 40. The creation of DIS-XML is achieved through the binding IEEE DIS to XML through Java Objects.

The XML Schema for IEEE DIS PDUs is not a standard schema, but it is hoped that other groups will collaborate, contribute, and ultimately agree upon a standard for this information set conversion. Similarly, there are not standardized methods for language binding of IEEE DIS. Nevertheless, availing DIS in XML format will increase the potential uses for the data.

Once represented as XML, DIS data is a candidate for transport over XMPP. This possibility is attractive because IEEE DIS multicast traffic is typically disallowed over routed and switched networks. Passing DIS-XML over XMPP is one way to provide distributed DIS simulation across the Internet.

D. XTC CHAT LOGGING

Logging, or archiving, of chat messages is a desirable feature for many individuals and organizations. The military, in particular, has great need for data collection of message traffic. Any military implementation of XMPP chat systems will require message logging.

Many XMPP server implementations provide a configurable message logging feature, as seen with Wildfire in Chapter V. Additionally, as seen with the Transverse

client, there are XMPP chat clients that permit local logging of chat conversations. It would be desirable, however, to enable the storage and retrieval of chat message data on an enterprise level. This area of XTC research seeks to design and demonstrate an example of such a feature.

XTC Chat Logging uses client applications, written with the Smack API, to listen for XMPP message stanzas and writes their content to an XML file. These log files are then periodically collected into a single Native XML database, eXist, and exposed through a web-application, written in XQuery, for query/search. Figure 41 diagrams this process.

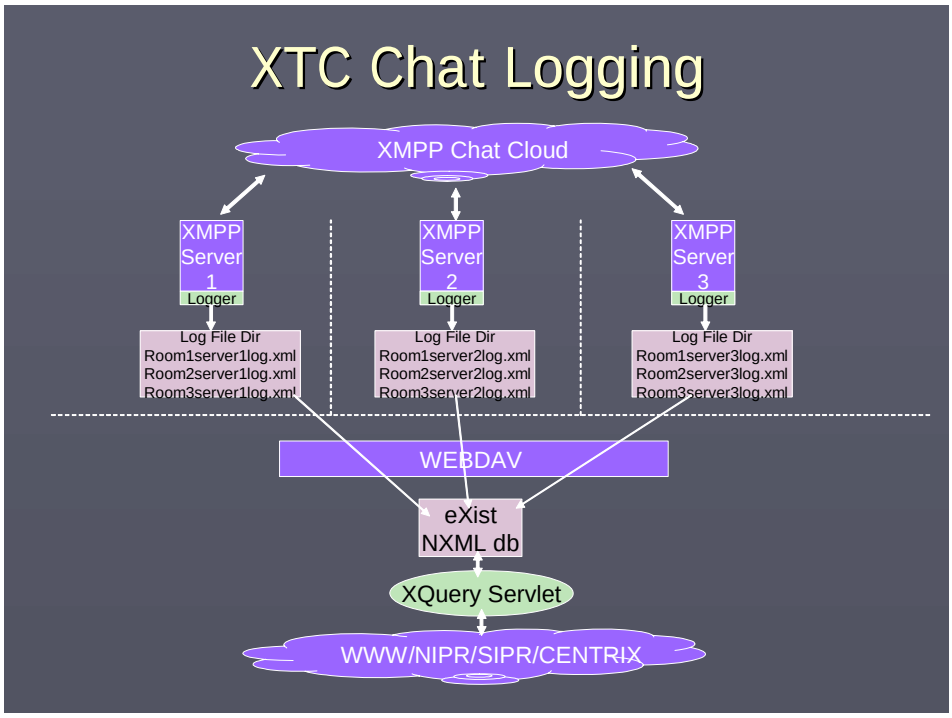

Figure 41. The XTC Chat Logging Architecture writes chat conversations to XML files and consolidates them into a web-browser accessible data base.

There are many potential implementations of an enterprise storage and retrieval service for chat data. The XTC Chat Logging model presents one possible solution with design emphasis on a few decision points: retention of data as XML throughout the system, periodic collection of files into an enterprise database rather than direct collection of chat messages, and exposure of chat data for query/search through a web application. The desire for retention of XML format is to eliminate the problems associated with data-

format conversions. The desire for periodic consolidation of data files is motivated by the potential for frequent disconnection and reconnection of the database from the supported XMPP servers due to disruptions in the tactical military networks. Finally, the use of a web-application facilitates access to the chat data by eliminating the need for additional tools or knowledge of query languages.

E. JC3IEDM-ENHANCED TACTICAL COLLABORATION (JTC)

Led by Naval Undersea Warfare Center in Newport, Rhode Island, JC3IEDM-Enhanced Tactical Collaboration (JTC) is a command and control experiment conducted as part of Trident Warrior 2006. JTC effort included construction of the JTC Chart/Map application, which provides interactive charts, maps, and forms, and Chat/IM to aid users in performing planning, tasking, reporting, and information querying tasks. JTC uses XML exposed JC3IEDM business objects that describe a set of military operations. These objects are used to create interactive planning tools that, along with chat discussion, seek to enable more efficient collaborative task planning than working with slide presentations and e-mail. Figure 42 provides an example of the user interface of the JTC Chart/Map application (Chaum, 2006a).

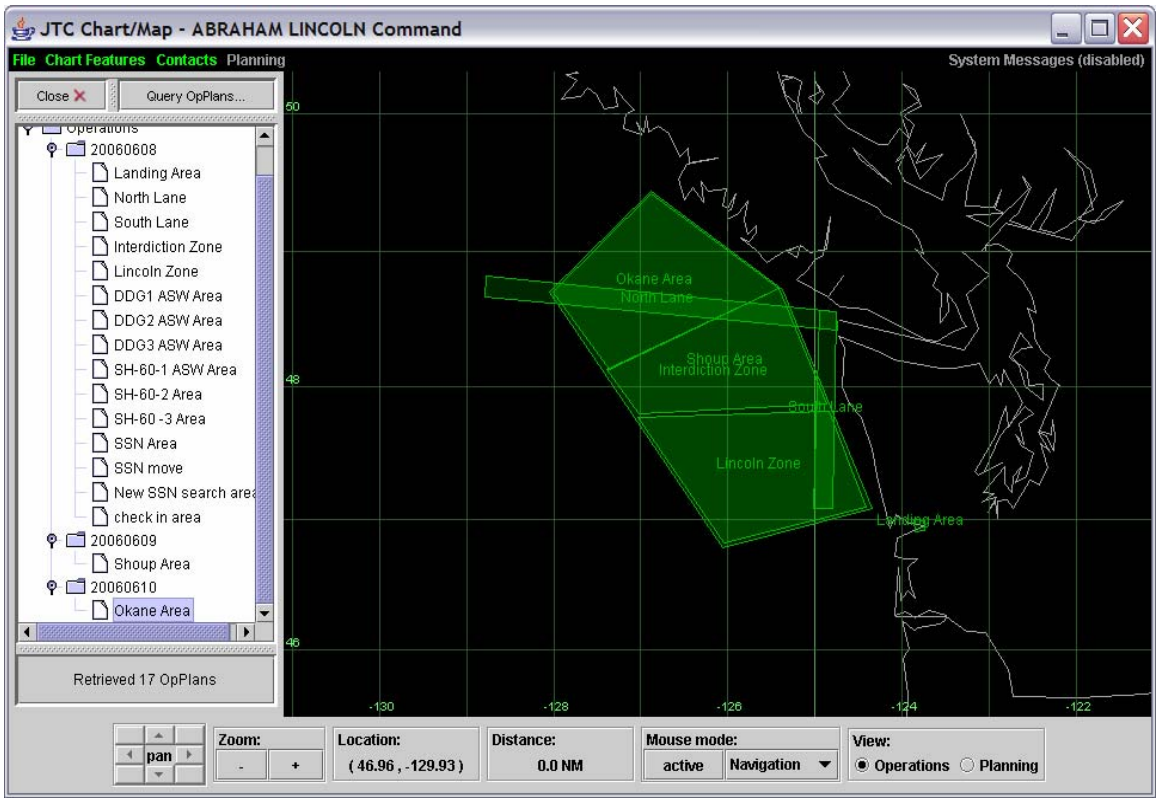

Figure 42. The JTC Chart/Map provides a Graphic User Interface (GUI) for common situational awareness and maritime operational task creation and presentation.

As indicated in Figure 43, the JTC architecture uses a relational data store to maintain the JC3IEDM tasking objects and uses an API to convert these tasks into XML messages to be passed over the XMPP network. At the receiving end, these objects are passed to the Chart/Map application and processed, rendering a graphic overlay or updating some associated data. These tasks can be created or modified through either individual or collaborative effort and subsequent published to the JC3IEDM data store (Chaum, 2006a).

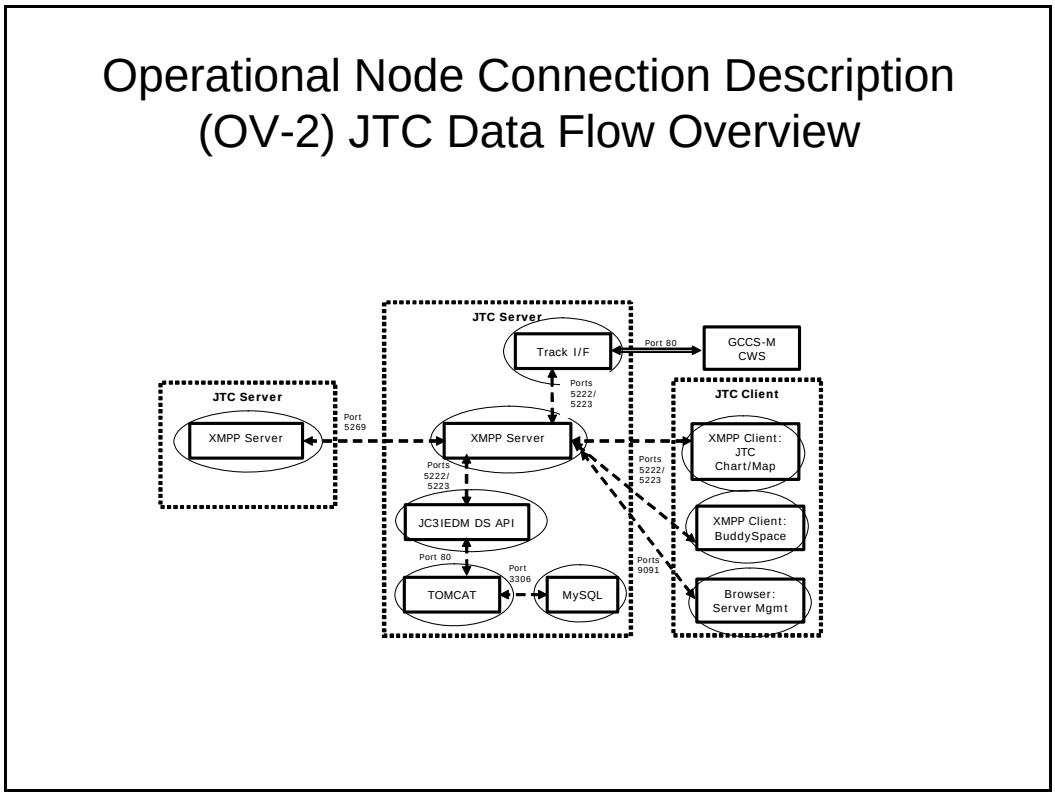

Figure 43. The JTC Data Flow Overview describes the flow of information during JTC Trident Warrior 2006.

JTC is an example of how XMPP can be employed to route not only human-to-human chat traffic, but also machine-to-machine purposed XML data, on a single network. JTC uses the XMPP network for both purposes, demonstrating the value of XMPP as a general purpose XML routing network.

JTC was tested as part of the Navy's annual Trident Warrior 2006 (TW06) Exercise. Application functionality test preparations and exercise conduct are described further in Chapter VII.

F. SUMMARY

This chapter discussed the XML Tactical Chat project and presented some of the research focus areas. XTC research includes: using the XTC architecture to participate in the testing of XMPP chat over military tactical network systems, employing XMPP for the routing and delivery of DIS-XML data in support of military modeling and simulation, design, implementation, and testing of an XMPP chat logging system to

support military organizations, and participation in the JC3IEDM-Enhanced Tactical Collaboration experiment as part of Exercise Trident Warrior 2006. Chapter VII presents the results of this research.

VII. APPLICATIONS AND EXPERIMENTAL RESULTS

A. INTRODUCTION

This chapter discusses the results of XTC research experiments and provide a presentation of XTC related applications. NPS participated in two large military experiments through XTC, the CCSG12 XML Tactical Chat Test and JTC / Trident Warrior 2006. XTC related applications presented in this chapter include: the Autonomous Underwater Workbench (AUVW), a simulator for underwater robots that has been extended to operate with DIS-XML, and the XTC Chat Logger, a chat logging system that was applied and extended in support of the JTC / Trident Warrior 2006 exercise.

B. COMCARSTKGRU12 (CCSG12) XML TACTICAL CHAT TEST

In October 2005, Commander Carrier Strike Group 12, through Communications Officer, Commander Danelle Barrett USN, coordinated a combined fleet, joint, and coalition test of XMPP chat communications. Motivated by the lack of interoperable Chat/IM communications in the military community and the security inadequacies of many deployed chat systems, CCSG12 secured test participation from a large number of organizations. The below listed units participated and/or assisted in the test (Barrett, 2006).

- U.S. Joint Forces Command (USJFCOM) – Norfolk, Virginia
- U.S. Pacific Command (USPACOM), Commander, Pacific Fleet (CPF), Pearl Harbor, Hawaii
- Defense Information Systems Agency (DISA)
- NATO Supreme Allied Command Transformation
- U.S. Air Force Command and Control, Intelligence, Surveillance and Reconnaissance Center (AFC21SRC) – Langley, Virginia
- Naval Postgraduate School (NPS) – Monterey, California and a remote user in Arlington, Virginia
- Space and Naval Warfare Systems Command (SPAWAR) – San Diego, California

Figure 44 depicts an overall schematic of the test.

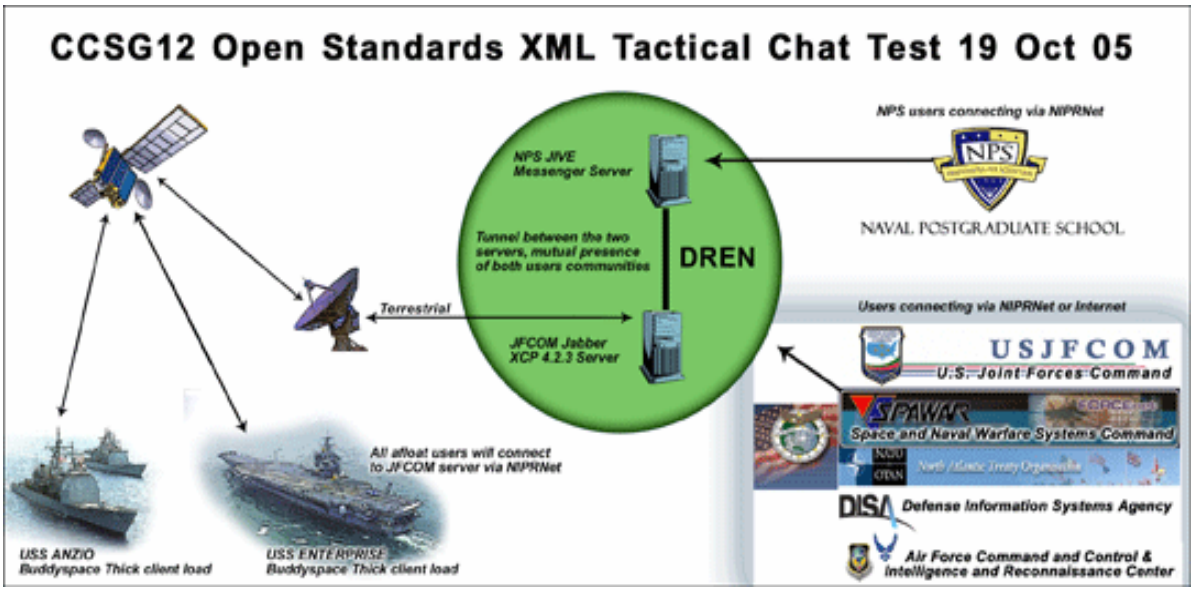

Figure 44. The CCSG12 XML Tactical Chat Test of October 2005, demonstrated the feasibility of XMPP chat over ship borne satellite communications.

Generally, this test sought to evaluate the suitability of XMPP chat communications for military use and to assess the degree of interoperability of a set of XMPP tools. The following were identified as test objectives:(Barrett, 2006)

- Connect and federate the Jabber Jive and Jabber XCP 4.2.5 servers at NPS and USJFCOM respectively, and ensure presence of users and persistence between users on both servers.
- Load and test different **XMPP** compliant chat clients at several joint and coalition commands, including units at sea. The mix of clients needed to include thick (client/server) and Web-based clients. Interoperability out of the box among the various clients had to be verified.
- Hold a chat session with all participants for approximately two hours. Monitor bandwidth for afloat connections and other locations where data could be collected. Analyze bandwidth data to determine functionality of clients in a bandwidth disadvantaged environment, specifically on both large and small ships at sea.
- Collect subjective data from users about the functionality and performance of the different **XMPP** client types to determine acceptable and unacceptable user experiences.

In order to assess the degree to which these objectives were met, the following performance metrics for success were identified:(Barrett, 2006)

- Shipboard bandwidth utilization does not increase significantly (more than five percent).
- A minimum of two **XMPP** compliant chat tools successfully interoperate.
- Chat clients afloat are able to easily connect and function with server ashore.
- Chat tool is available 100 percent of the time during the test period assuming a stable satellite link.
- Chat tool is user friendly and intuitive for operators (assessed using a survey).
- Two standards-compliant chat servers are connected with presence of users established.

Figure 45 diagrams the test topology.

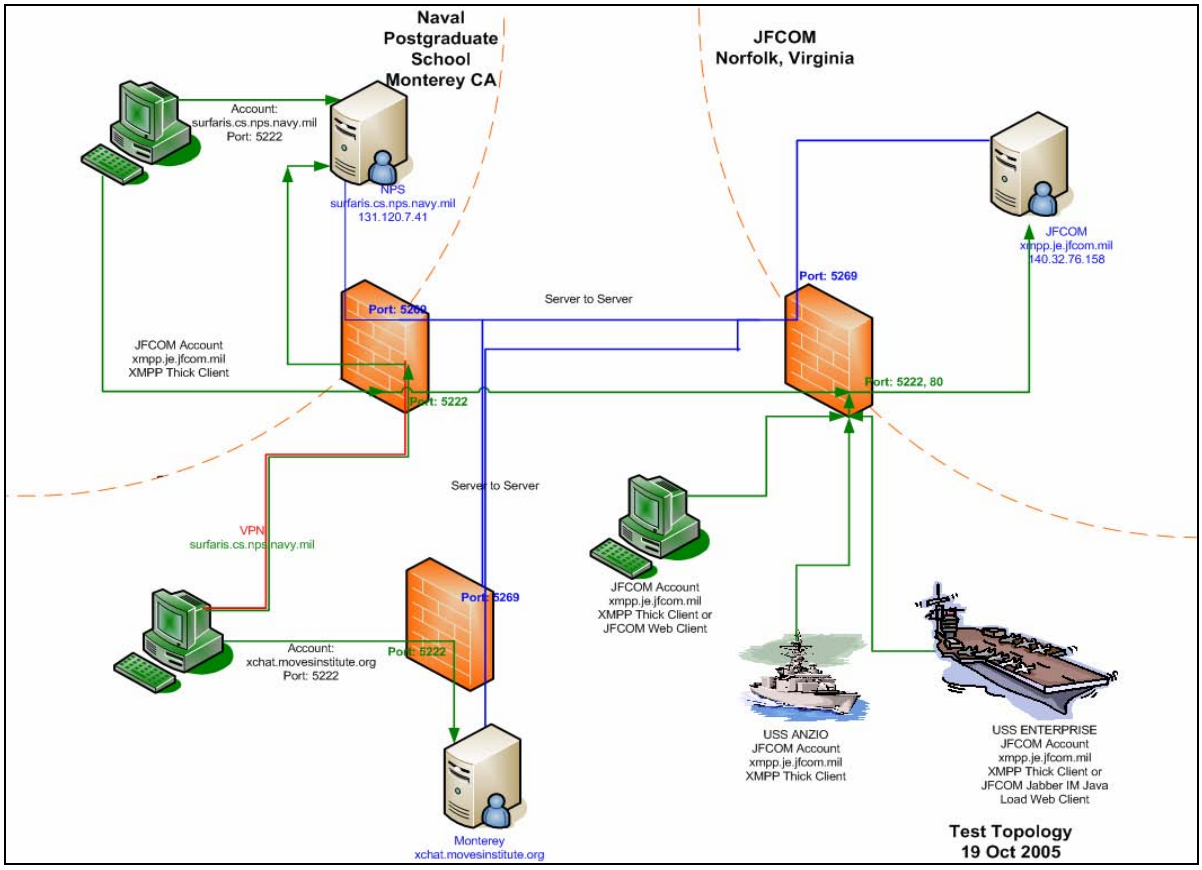

Figure 45. CCSG12 XTC Test Topology describes the network connections used in support of the CCSG12 Chat Test exercise.

The test required each of the participants joining a chat room be hosted on the `xmpp.je.jfcom.mil server`. Most users joined this room while connected to the

same server, though NPS participants logged into their local server and joined the test room through the server-to-server link. Once the participants were gathered, scripted text-traffic narratives were passed into the room, throughput data was collected on the ships, and the users evaluated XMPP client performance during the test. A voice line was also provided via satellite downlink to USS ENTERPRISE allowing additional exercise coordination ensuring all chat-channel tests proceeded smoothly. The test period was approximately 90 minutes.

The findings from the test are listed below: (Barrett, 2006)

- The user experience on the ships with the XMPP client was as good or better than that with the currently deployed chat tools.
- The bandwidth consumption of the test was supportable by the ship borne communications systems and is comparable to that of other chat/IM systems.
- Estimated user bandwidth requirements were 0.11 KBps for passive users and 0.13 KBps for active chat users.
- XMPP thick-client applications were both more reliable and more bandwidth efficient than the HTTP polling web-based clients.
- Persistent room history is an important feature for shipborne chat, as it allowed disconnected users to recapture conversation context upon reentering the chat room.
- Federation of XMPP servers was effective in demonstrating the distributed, federated chat architecture that will need to be implemented in the military.

The performance of the web-based XMPP chat client was a disappointment of the test, however, it should be noted that this client employed the HTTP polling method of server access over port 80. The JEP supporting HTTP polling (JEP-0025) has been deprecated and replaced with JEP-0124: HTTP Binding. Given the performance requirements listed in JEP-0124, it is likely that a well-designed implementation of this standard will be more responsive and reliable than the web client employed in this test (Smith, Saint-Andre, and Paterson, 2005).

Encouraged by the test findings, CCSG12 presented the following recommendations: (Barrett, 2006)

- Commander, Naval Network Warfare Command (NETWARCOM) consider a policy making XMPP the approved open standards chat protocol for the fleet and shore Navy, and approve XMPP port use through the fleet firewalls and proxy servers.
- Navy FORCEnet and SYSCOM engineers develop a consolidated plan to implement a distributed, federated, XMPP compliant chat solution for the fleet and eliminate non-XMPP chat programs. Each ship should have its own XMPP chat server so it can continue operations internally during periods when disconnected from the satellite link. Replication and synchronization of chat server data should be carefully engineered.
- Navy FORCEnet and SYSCOM engineers should leverage work done by LAPS and USFJCOM to apply compression algorithms to XML chat, which will improve bandwidth efficiencies afloat. Current research and testing achieves XML chat compression by a ratio of 3:1 without noticeably increasing latency of the chat session.
- NETWARCOM work with the SYSCOMs to collectively consider using Transverse, the open standard, open source freeware developed by USJFCOM based on the Jabber Instant Messaging model as the software for afloat forces.
- Navy representatives to the DISA Global Information Grid (GIG) Net-Centric Enterprise Services (LACES) Working Group support only XML compliant, bandwidth friendly solutions for the follow-on to DOTS.
- Continue to test XMPP and other open standards compliant collaborative tools in a joint, coalition and interagency environment. The continued development of joint capabilities around open standards should drive Navy solutions particularly when the Navy doesn't have an existing capability.
- Consider XMPP chat and all collaborative tools as enterprise services. Ensure the Navy Marine Corps Intranet (NMCI) adopts XMPP as its instant messaging and text chat solution and that an improved XMPP client be installed on all NMCI workstations. This is particularly important for embarkable staffs moving between the NMCI and afloat network enclaves.
- As the Navy continues to put into place key components of the FORCEnet architecture, adherence to open standards collaborative tools, such as those tested during this exercise, will ensure maximum interoperability in future warfighting, peacekeeping and humanitarian relief operations.

Appendix B is a copy of the post-exercise message released by COMCARSTRGRU 12, describing exercise details and recommendations.

C. DIS-XML / AUV WORKBENCH

XTC research with DIS-XML sought to achieve two things: to demonstrate the routing of DIS-XML over an XMPP network to the effect of distributing simulations across networks, and to import DIS-XML format into the AUV Workbench.

To achieve the first objective, two Java applications, utilizing the `org.web3d.xmsf.dis`, `org.web3d.xmsf.disutil`, `org.jivesoftware.smack`, and `org.jivesoftware.smackx` libraries, were written. The first, `XMPPSender.java`, establishes a connection to an XMPP server, logs in, and joins a chat-room. It then enters a loop that instantiates Java DIS PDU objects, marshals them to XML, attaches the DIS-XML as a Java String property value to a Smack supported XMPP message, and, finally, sends these XMPP messages into the chat room.

Figure 46 is a sample message sent by XMPPSender object, logged into the XMPP server as "snerd0."

```
<message id="qoU4g-67" to="adarmold@surfaris.cs.nps.navy.mil/Exodus" type="groupchat"  from="disxml@conference.surfaris.cs.nps.navy.mil/snerd0">
    <body>A DISXML message from the sender.</body>
      <properties xmlns="http://www.jivesoftware.com/xmlns/xmpp/properties">
        <property>
          <name>disXML</name>
          <value type="string"><?xml version="1.0" encoding="UTF-8"
      standalone="yes"?>
                    <DIS>
                    <EntityStatePdu capabilities="0" entityAppearance="0" forceID="0"
    numberOfArticulationParameters="0">
                    <PduHeader length="144" pduType="1" protocolFamily="1" timestamp="0"/>
                    <EntityID application="1" entity="2" site="0"/>
                    <Entity/>
                    <AlternativeEntity/>
                    <EntityLinearVelocity/>
                    <EntityLocation x="62.0"/>
                    <EntityOrientation/>
                    <DeadReckoningParameters otherParameters="00000000000000000000000000000000">
                    <EntityLinearAcceleration/>
                    <EntityAngularVelocity/>
                    </DeadReckoningParameters>
                    <EntityMarking characterSet="0" characters="00000000000000000000"/>
                    </EntityStatePdu>
                    </DIS>
            </value>
        </property>
    </properties>
</message>
```

Figure 46. `XMPPSender.java` sends DIS-XML <message/> stanzas such as this. Note the XMPP value element ultimately contains DIS-XML data as its payload value.

The second application, `XMPPReceiver.java`, also logs into the XMPP server and joins the same chat room. Once in the room, the application listens for message traffic that is labeled as containing DIS-XML data as a property, creates a Java string and assigns the value of the DIS-XML data, and un-marshals the XML into a Java DIS PDU object. This object is then marshaled into native IEEE DIS form as a byte stream and sent out as a multi-cast packet(s) onto a local DIS network. A DIS-compatible viewer is then used to visualize the behavior of the simulation object.

Figure 47 captures the chat room and viewer behavior prior with only the receiving bot (snerd1) and the author in the chat room. Figure 48 captures the same behaviors after the sending software bot (snerd0) enters the chat room and is sending its messages with DIS-XML payload.

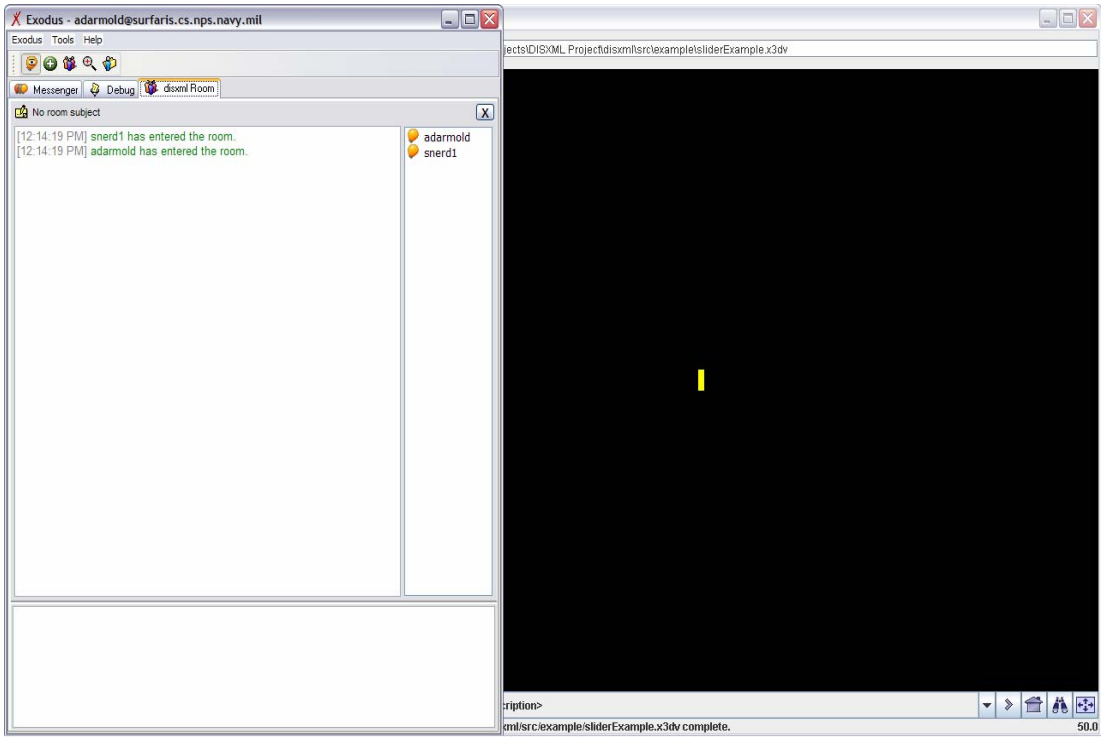

Figure 47. XMPP MUC Room (left) and Xj3D Browser (right) with `XMPPReceiver.java` running and waiting for messages with DIS-XML payload.

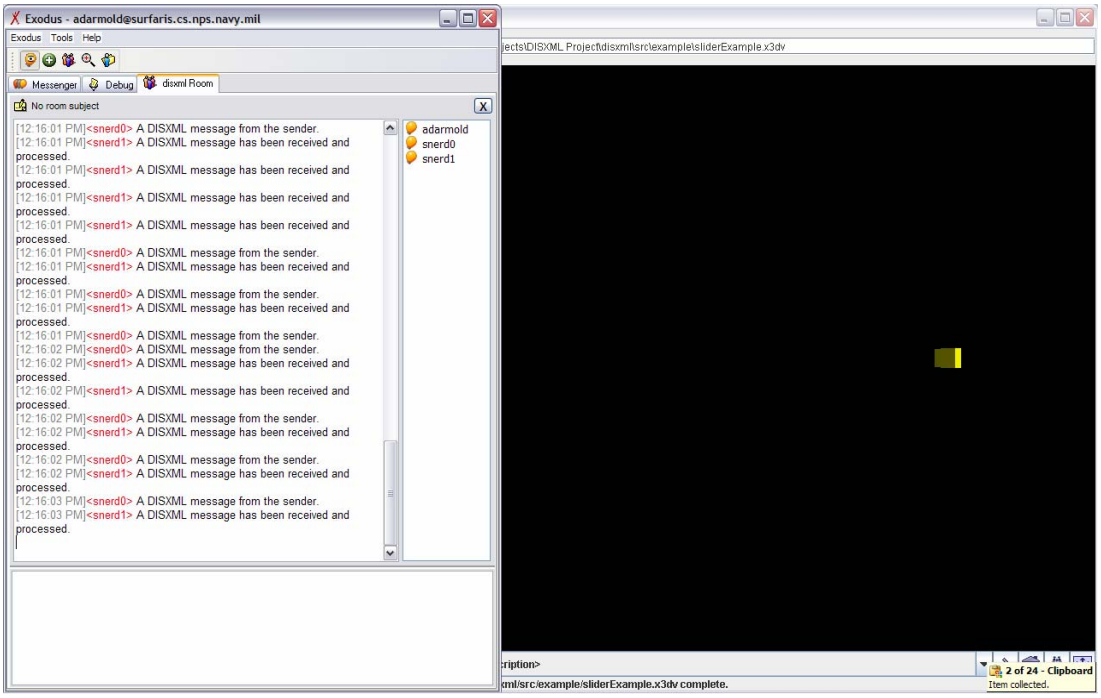

Figure 48. XMPP MUC Room (left) and Xj3D Browser (right) with `XMPPSender.java` sending XMPP messages, `XMPPReceiver.java` processing these messages, and the Xj3D Browser exhibiting DIS entity movement.

Once it was demonstrated that the XTC XMPP network might be purposed to route DIS-XML using these messages to drive DIS simulation, it was desirable to import this function into a suitable simulation tool. The AUV Workbench is a tool that allows for the simulation of autonomous underwater vehicle (AUV) activity. AUV's are expensive to build and problems experienced with them during testing and missions are difficult and expensive to recover from. The AUV Workbench provides a platform with which to test, with high environmental fidelity, vehicle models, mission sequences, and other use cases, without the risk of damaging or losing the vehicles themselves. The AUV workbench already included native DIS support.

The AUV Workbench was extended to provide DIS-XML and chat-enabled transport. This feature provides the ability to perform simulations between multiple network nodes, even when there is no multicast support between these nodes. To further support such distributed simulation activity, an XMPP chat console was also added to the AUV Workbench. This feature provides login access into a chat server, entry into a chat

room, and the basic GUI interface features to send and receive chat messages within the AUV Workbench window. These features are displayed in Figure 49. With the addition of these features and an XMPP network, distributed AUV simulations have been conducted through firewalls and across the internet. This is an exciting advancement in the area of modeling and simulation, and an example of the extensible power of XMPP.

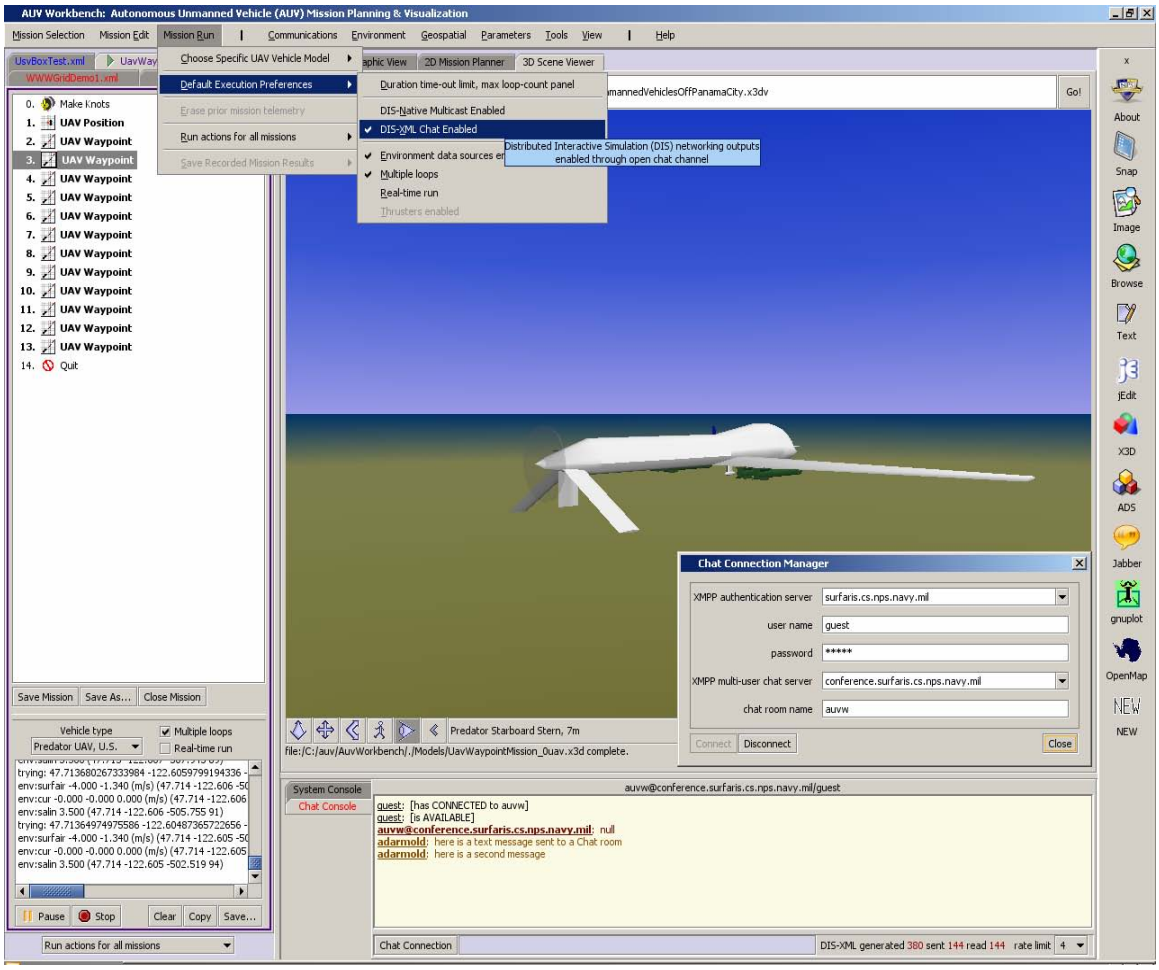

Figure 49. Autonomous Underwater Vehicle Workbench. Note the ability to select DIS-XML as an execution format and the inclusion of an XMPP chat console as simulator features.

Future work in this area includes the integration of XML Namespace support (W3C 1997) to properly distinguish the integration of separate XML targets for XMPP and DIS-XML.

D. XTC CHAT LOGGER

As discussed in Chapter VI, the XTC Chat Logger is an example solution for enterprise wide chat message logging, storage, and retrieval. The XTC Chat Logger consists of three components: `ChatMessageLogger.java`, a XMPP client Java application, eXist, a Native XML Database, and `xtclog.xql`, a web application written in XQuery.

`ChatMessageLogger.java` uses the `org.jivesoftware.smack` and `org.jivesoftware.smackx` libraries for XMPP support. The application acts as a XMPP client, The use of a client logger, however, does prevent the logging of instant messages and private-chat rooms. Though plug-in support for Smack does not exist on the Wildfire servers, a server plug-in for logging support could be created, allowing for comprehensive logging of data. Connection, login, and MUC room joining settings are established with an accompanying properties file, named `xmpp.properties`. This file contains a list of all MUC rooms that should not be logged. Once logged into the server, the application joins all hosted MUC rooms with the exception of those on the "doNotLog" property list. The application then listens for all <message/> stanzas that are sent to the MUC rooms, captures the <body/> elements and other data, and writes them into a well-formed XML file. The application also checks all messages for a "jabber:x:delay" namespaced <x/> element. This extension element is the recommended practice for time-stamping message delay. `ChatMessageLogger.java` adds "jabber:x:delay" time-stamp data to all messages not previously stamped.

Figure 50 captures the processes of `ChatMessageLogger.java`.

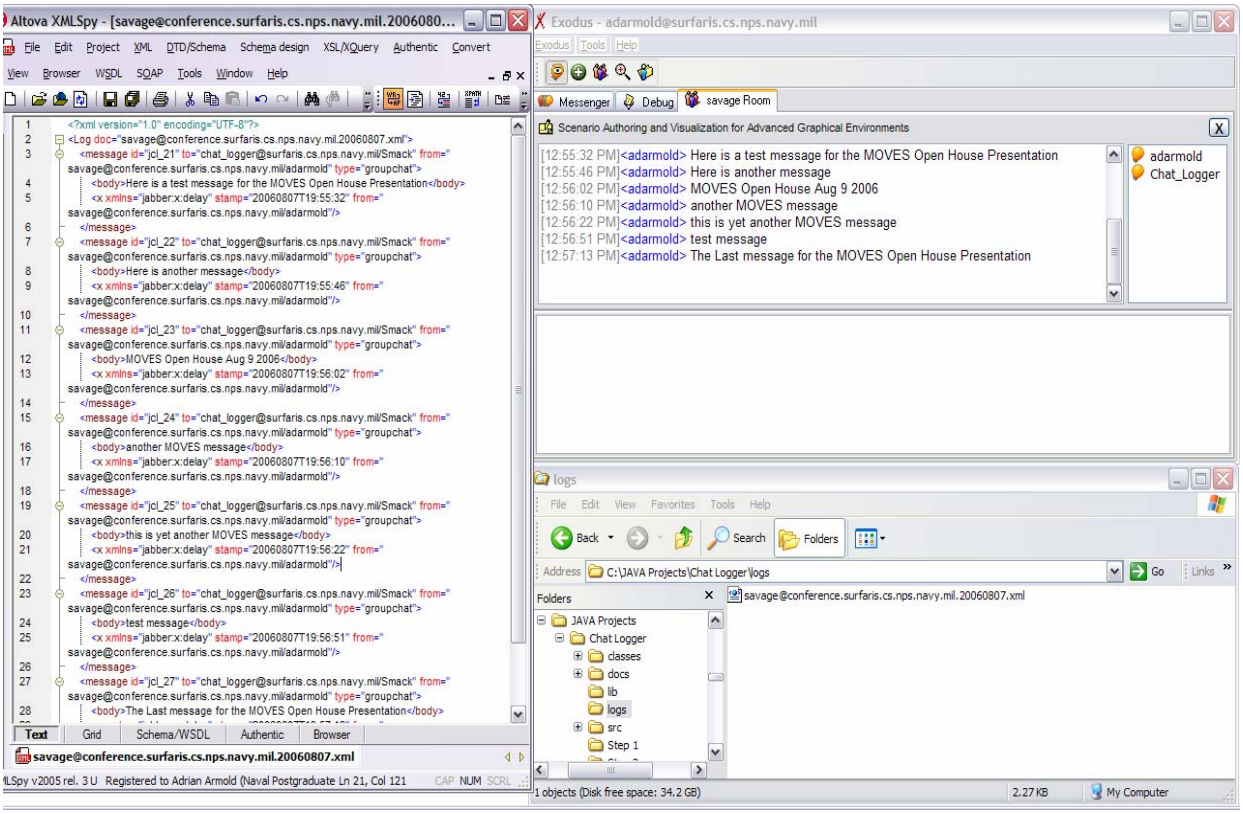

Figure 50. Screen capture of `ChatMessageLogger.java` processes. MUC room messages are captured in a chat room (top right) and written to a well-formed XML file (lower right). The contents of the file are seen at left.

At the time of writing, these files are manually imported in an eXist Native XML database. This is performed via a simple Web-based Distributed Authoring and Versioning (WebDAV) enabled operation. eXists's use of WebDAV allow for a viewable abstraction of the database as a file directory as seen in Figure 51.

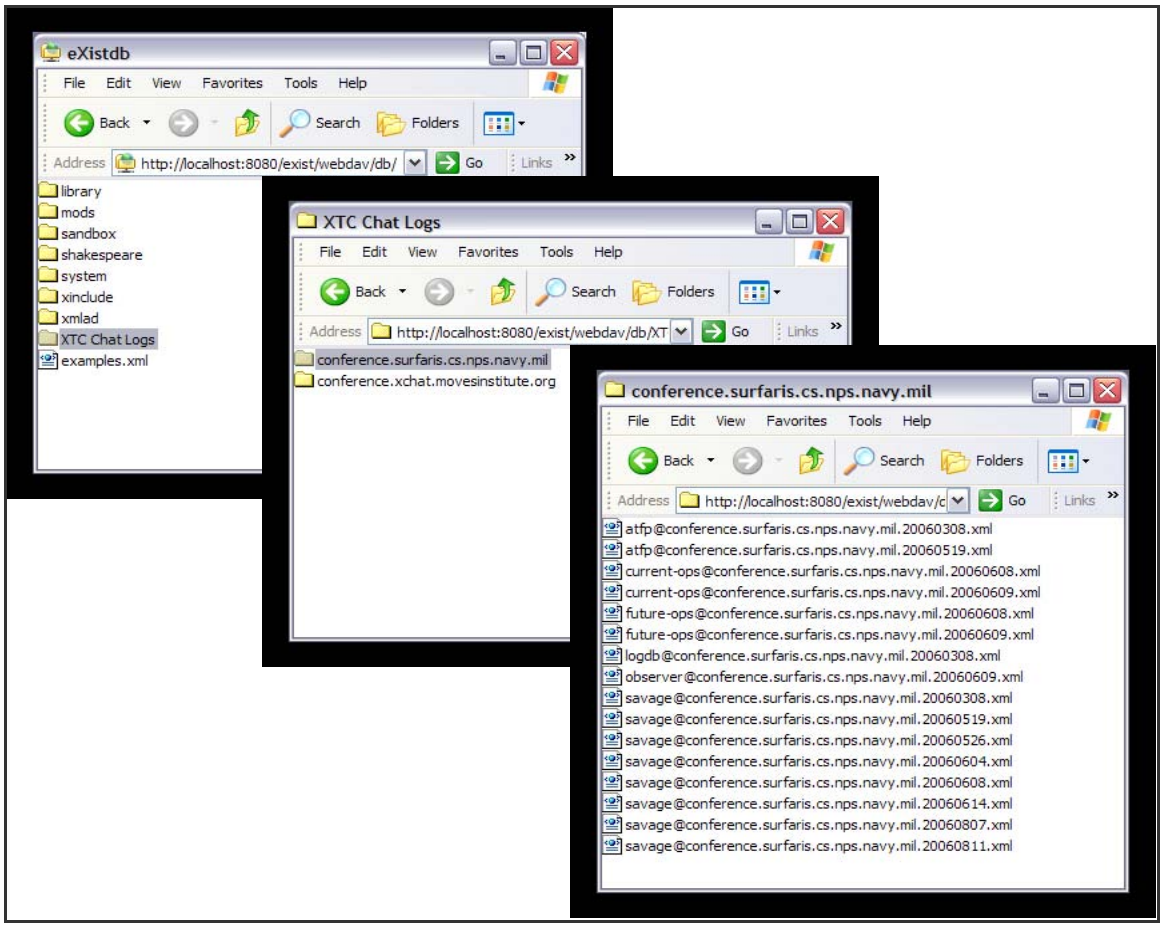

Figure 51. The eXist database is WebDAV enabled, permitting the data to be viewed as a file hierarchy. WebDAV also facilitates the importation of XML documents into the database.

The last component of the XTC Chat Logger is the web application, providing text search on the chat logs. This application, xtclog.xql, was written in XQuery and deployed as a web-app within the eXist database, itself deployed as a web-app in a Tomcat web server. Figures 52 and 53 capture the function of xtclog.xql; a keyword is entered as input, all chat messages whose <body/> contains the keyword are returned. These results are returned as hypertext, pointing to the full chat log they were drawn from.

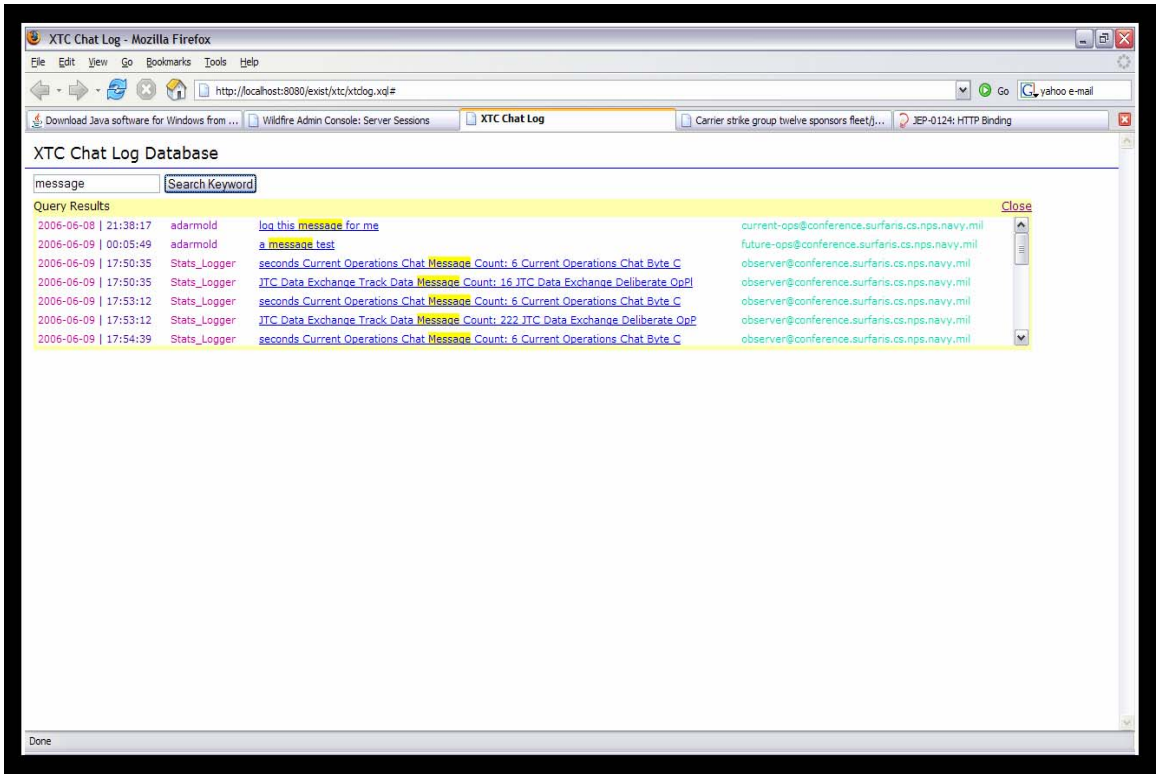

Figure 52. XTC Chat Logger provides keyword search on stored chat logs.

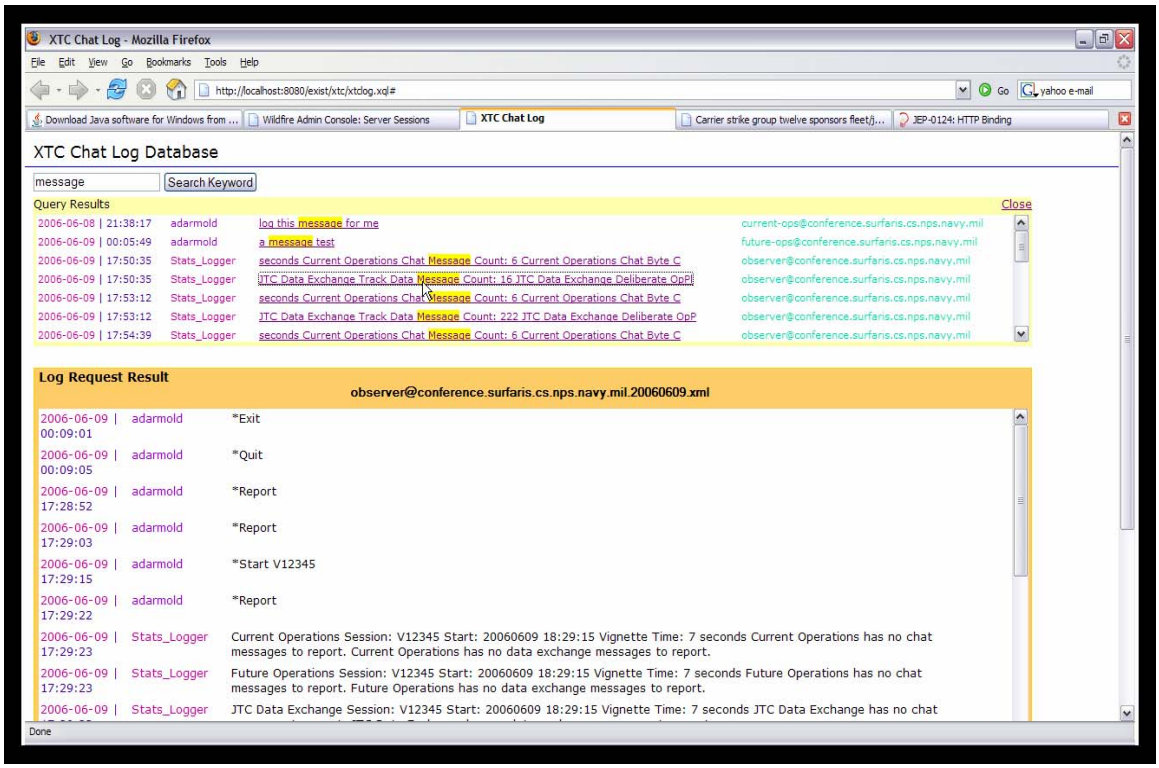

Figure 53. Keyword search results are hypertext message bodies that point to their chat log of origin.

The XTC Chat Logger is a demonstration of the value of using XML formatted chat data. It is a simply designed, easily implemented tool that required only moderate programming expertise and a small amount of code to produce. Interested readers can refer to Appendix A for guide for installing and running the XTC Chat Logger. Future work includes exposing this application as a web-accessible servlet running on a chat-connected host.

E. JTC / TRIDENT WARRIOR 2006

1. Trident Warrior Experiment Series

Trident Warrior is a series of at-sea exercises designed to explore command and control capabilities, along with associated Tactics, Techniques, and Procedures (TTPs), to optimize naval warfighting operations. Trident Warrior 2006 sought to focus on these capabilities with respect to operating in the joint and coalition operational environment (Woods, 2005).

2. JC3IEDM-Enhanced Tactical Collaboration Experiment

As part of Trident Warrior 2006 and sponsored by the Office of the Secretary of Defense, Acquisitions, Technology, and Logistics (OSD-ATL), the Naval Undersea Warfare Center (NUWC), and the Naval Postgraduate School conducted an experiment called JC3IEDM-Enhanced Tactical Collaboration (JTC). JTC sought to create and test an operational planning system that could more rapidly and efficiently support mission planning, tasking, and querying than existing e-mail and slide presentation methodologies. JTC is comprised of a planning tool application that uses interactive maps, charts, and forms to support collaborative planning sessions and present JC3IEDM based Common Operating Picture (COP) awareness and chat messaging to support collaborative planning discussion. As discussed in Chapter VI, the JC3IEDM COP messages are XML formatted. Both the human chat messages and the machine-purposed JC3IEDM messages are routed over an XMPP network.

3. JTC Operational Threads

The JTC experiment sought to explore the efficacy of using structured data, available for interactive collaboration and passed in real-time for executing operational planning tasks. Some tasks required the use of deliberate planning data elements while others required collaborative planning data elements. The operational tasking threads that

were tested are depicted in Figure 54. Although the necessary functionality was implemented, the inclusion of live track data from a GCCS-M C4I Web Service feed was not able to be tested due to the lack of an available service site (Chaum, 2006b).

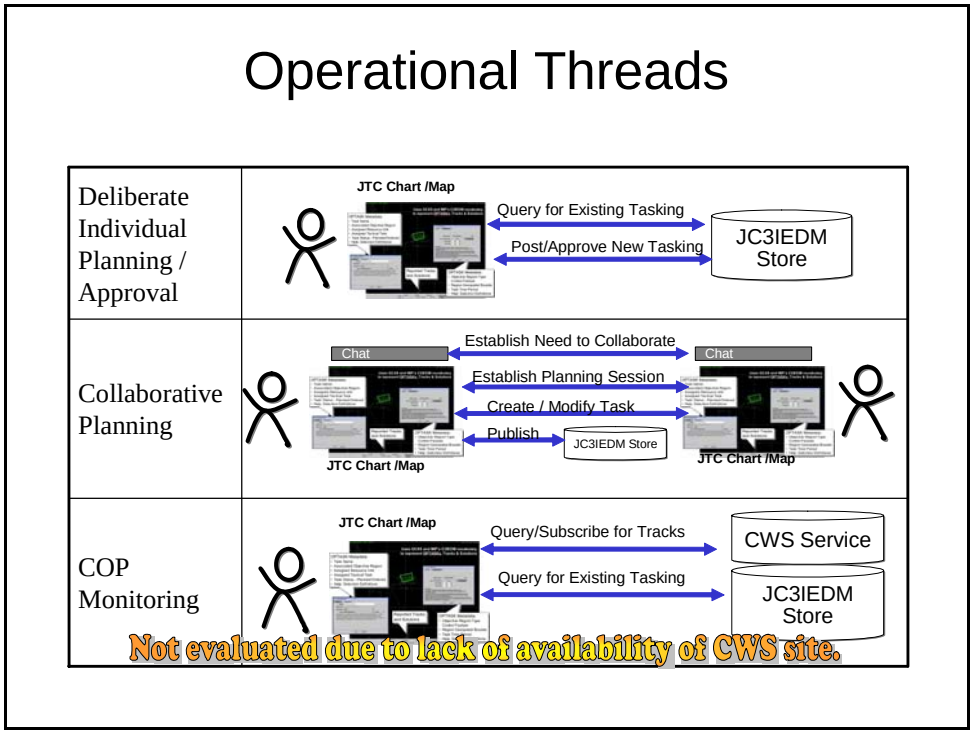

Figure 54. Two operational threads were evaluated in the JTC Trident Warrior 2006 experiment. Note that COP monitoring was not able to be tested as no C4I Web Service site was available.

4. Mission Vignettes

The planning tasks tested were maritime operational missions for Strike, Mine and Inshore Warfare, Maritime Interdiction Operations, and Anti-submarine Warfare. With JTC, the operational and tactical concepts and semantics of these missions were expressed through JC3IEDM XML message templates. Figures 55 and 56 display examples of the JTC mission vignettes executed during the experiment (Chaum, 2006b).

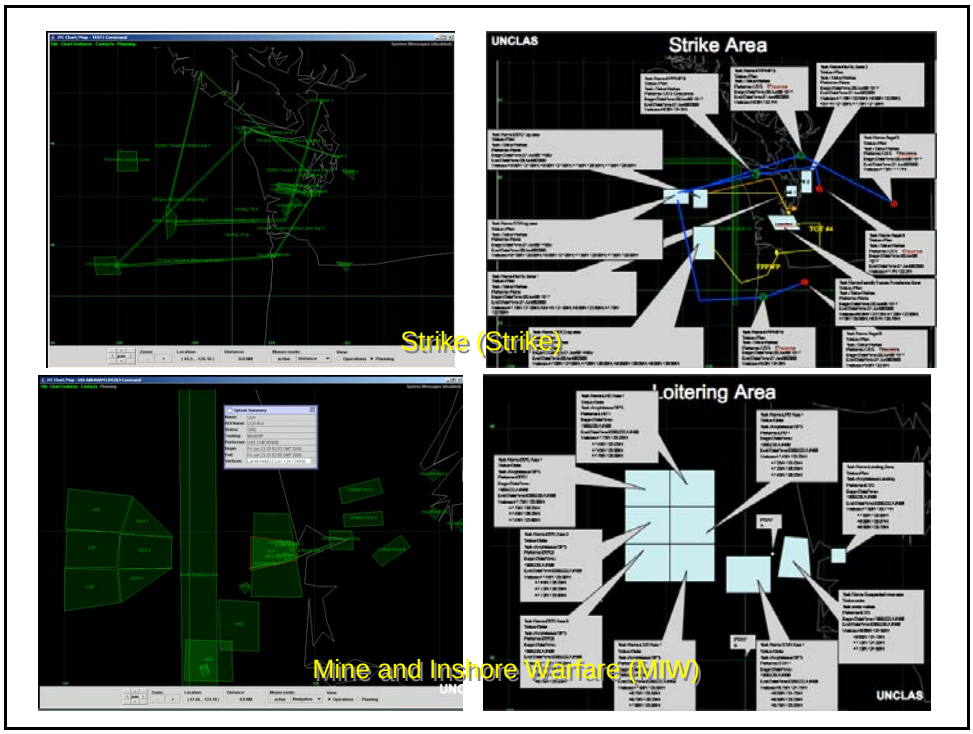

Figure 55. Strike and Mine and Inshore Warfare mission vignettes from the JTC Trident Warrior 2006 experiment.

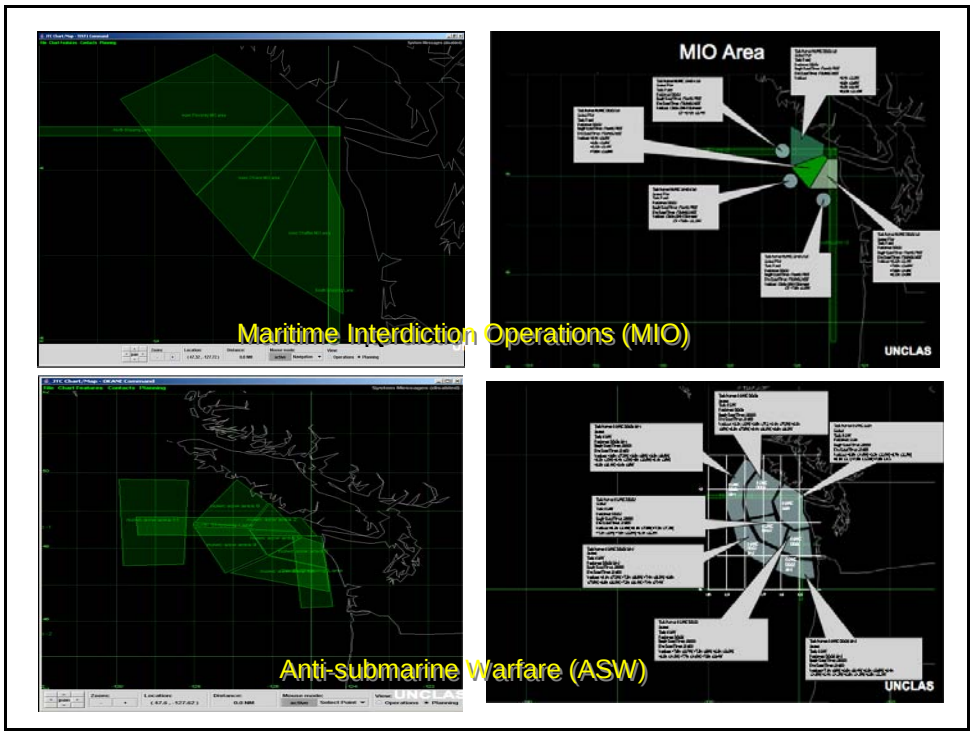

Figure 56. Maritime Interdiction Operations and Anti-Submarine mission vignettes from the JTC Trident Warrior 2006 experiment.

The vignettes were used to create an operational scenario in which a user would be required to perform a planning task. Typically, a vignette would be comprised of a scenario description, a list of initial tasks or planning information required to perform the tasks, and a list planning tasks and objects to be completed and created. Users were then given specification on which tools should be used to create the planning objects. While users completed the vignettes, data was collected on the time required to complete the tasks and on the size of the planning objects created to complete the tasks. Appendix C describes the JTC vignettes in greater detail.

5. JTC Architecture

JTC tested user/actor performance in completing mission planning tasks using the JTC tools against performing those same tasks using Power Point presentations and email. The experiment for Trident Warrior 2006 involved individuals working in four locations, NUWC, in Newport, Rhode Island, NPS, in Monterey, California, Commander Pacific Fleet, in Pearl Harbor, Hawaii, and the USS Bonhomme Richard (LHD-6) underway. The architecture for the JTC experiment for Trident Warrior 2006 is depicted in Figure 57 (Chaum, 2006b).

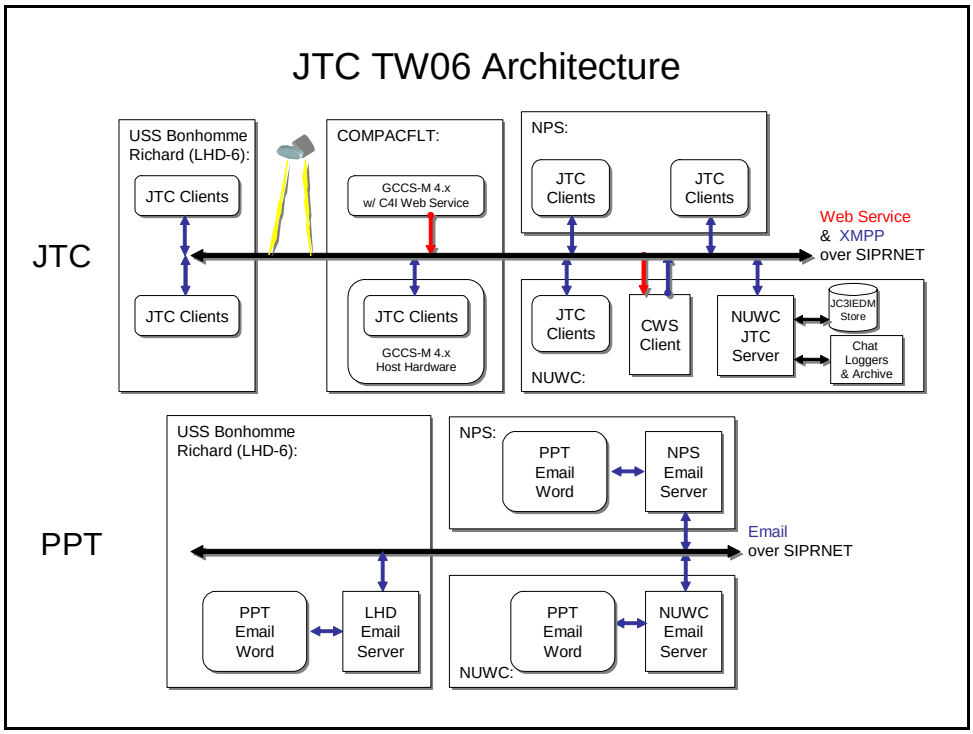

Figure 57. JTC Trident Warrior 2006 Architecture included users at COMPACFLT, NPS, and on the USS Bonhomme Richard.

The XMPP network hosted four functional chat rooms for the experiment. Two of these were for human actor use to discuss and coordinate planning tasks, one supported the machine-to-machine purposed JC3IEDM XML message data, and one served as an experiment control room. Table 12 describes the chat room support for JTC Trident Warrior 2006.

JID	Description	Purpose
`current-ops@conference.jtc.asw.navy.smil.mil`	JTC Current Operations Room	Supports chat discussion and coordination with regard to current operations.
`future-ops@conference.jtc.asw.navy.smil.mil`	JTC Future Operations Room	Supports chat discussion and coordination with regard to future operations.
`jtc-de@conference.jtc.asw.navy.smil.mil`	JTC Data Element Room	Supports machine-to-machine purposed JC3IEDM message traffic for JTC Chart/Map
`observer@conference.jtc.asw.navy.smil.mil`	JTC Observer Room	Supports JTC experiment coordination and supports command input for the experiment statistics collection application.

Table 12. Chat rooms used in support of the JTC Trident Warrior 2006 experiment.

6. JTC Statistics Logging

In addition to measuring the time required to perform the planning tasks with the two architectures, a logging tool was used to measure the bandwidth consumption of JTC traffic. A derivative of the Chat Message Logging program, `ChatStatsLogger.java` is an XMPP client that measures and reports on message traffic in specified chat rooms. It accepts text commands, in a specific room, that dictate the commencement and termination of statistics gathering and reporting on gathered statistics. This application records for a given session: start time, session length in

seconds, number of chat messages the room received, number of deliberate and collaborative data element messages the room received, number of chat bytes the room received, and the number of deliberate and collaborative data element bytes the room received. On command, the client displays the gathered statistics for the current or most recent session in the JTC Observer room. These reports as well as all other chat room traffic generated in the rooms listed in Table 12 were logged and archived using the `ChatMessageLogger.java` application.

7. Results and Observations

Preliminary analysis of measured data and emphatic feedback from several days of exercise effort revealed that time requirements for JTC OPTASK efforts were superior by a factor of five or greater. Reducing standalone and collaborative planning tasks, from 5-10 minutes to two minute or less, exemplifies a significant improvement in tactical effectiveness.

The preliminary measure of bandwidth performance for the JTC experiment is presented in Table 13 (Chaum, 2006b). Initial results indicate that the JTC system not only allowed the users to more rapidly execute their planning tasks than when using e-mail and Power Point, but that the use of the JC3IEDM XML messages was also considerably more efficient with regard to bandwidth required to complete the task.

JTC OPTASK	~ 14,000 bytes
PPT OPTASK	~ 80,000 bytes

Table 13. Estimated average size of OPTASK planning object during the JTC Trident Warrior 2006 experiment was smaller when using JTC messages than when using Power Point.

Based on the initial results and the feedback of the JTC users, NUWC presented a number of observations were made regarding the JTC experiment. These are listed below: (Chaum, 2006b)

- Observation: JTC experimentation provided evidence that significant improvements in the area of speed-of-command can be achieved through a novel real-time interactive sharing of structured data/information. JTC

enhanced immediacy, clarity and uniformity of person-to-person collaborative communications.

- Observation: JTC used JC3IEDM, an open system-independent international C2 information exchange standard, to represent a variety of maritime operations. JC3IEDM proved useful for enhancing person-to-person as well as person-to-machine communications. JC3IEDM is designed for automated machine-to-machine information exchanges. JTC achieves enhanced real-time collaborative planning using a combination of formal data sharing and chat.
- Observation: JTC provided a capability for agile, rapid and efficient maritime planning. JC3IEDM-based OPTASK messages were created for a variety of representative scenarios and found to be very suitable for maritime operations.
- Observation: JTC capabilities (chart, chat, database, GCCS-M interface, data loggers) were successfully hosted as clients in an XMPP environment. The JTC operations required little bandwidth.

In addition to identifying these observations, some insight into the experiment design itself was made. The manner in which the statistics were gathered created some difficulties in post-exercise analysis. Statistics were collected on entire sessions, without distinguishing the statistics between the multiple OPTASKS created in a given session. Further implementations of the statistics collection module will require modifying the application code to provide for OPTASK-specific statistics tracking. Another potential addition to JTC would be to add a native XML database that would serve as a repository for both the chat messages and the JC3IEDM OPTASK messages. Unlike the XTC Chat Logger system, the messages would be written directly into the database rather than collected into an XML file. This addition would provide near-real time query access to OPTASK status via the web.

The JC3IEDM OPTASK messages were sent as message body content into the jtc-de chat room. Content placed inside the <body/> element of XMPP <message/> stanzas is intended to be human read. A future improvement to JTC will be to move the OPTASK message content to a namespace qualified packet extension, along with registering a server handler for this namespace. This can decouple the machine purposed traffic from any messaging intended for human use, and is more in keeping with the XMPP standard.

The JTC Trident Warrior 2006 (TW06) experiment proved successful in its objective in demonstrating that using interactive planning objects to support operational mission planning allows for more rapid and efficient execution of those tasks. More importantly, however, JTC was built on an open architecture of XMPP and JC3IEDM. JTC showed that the JC3IEDM was capable of supporting maritime operational and tactical planning concepts and semantics to the extent required for the mission vignettes. Because the JC3IEDM describes a vastly larger set of military operational concepts , these successful result provide strong evidence of the extensibility of the JTC concept. The use of XMPP in such a context demonstrates the value of the protocol as a generic XML routing network.

F. SUMMARY

The XTC research project has worked on a number of experiments and applications. This chapter discussed the CCSG12 XML Tactical Chat Test, DIS-XML and the AUV Workbench, XTC Chat Logging, and JTC Trident Warrior 2006. The experiments were largely successful in demonstrating: the suitability of XMPP systems on bandwidth constrained Naval tactical networking systems, the extensibility of XMPP for supporting XML message routing for modeling and simulation, an enterprise solution to chat message logging, storage, and retrieval, and the extension of an XML expressed data model and XMPP to build a command and control tool that enhances operational performance. These experiments and applications all serve as examples of the functionality and extensibility of XMPP based chat systems in support of command and control.

VIII. CONCLUSIONS AND RECOMMENDATIONS

A. CONCLUSIONS

This thesis presents XMPP as a viable and recommended solution for DoD chat and instant messaging requirements. It does so by describing the needs and requirements of military chat, examining the XMPP protocol and enhancements, and by presenting some current XMPP chat implementations. Furthermore, experimentation and application development is presented as both validation of the suitability of XMPP and as example of the extensible power of the protocol and XML messaging.

The chat and IM requirements of the military are only recently becoming defined. However, it is clear that existing chat and IM solutions in the DoD do not meet many basic information-system requirements with regard to security and information management. Furthermore, currently adopted solutions have resulted in interoperability problems for the DoD in the chat and IM domain. XMPP-based chat/IM solutions are well suited to meet the current needs of the military and provide an open-standard basis for overcoming interoperability problems.

XMPP presents an effective solution for military chat and instant messaging needs, as well as a framework for routing XML data that can be purposed for Command and Control enhancement. The XMPP protocol standard defines a core specification that defines this generic streaming XML framework. The XMPP Instant Messaging and Presence specification and collection of Jabber Enhancement Proposals comprise a rich set of standards for the development of solutions to military chat, IM, and other Command and Control applications.

The myriad of existing server and client applications serve as examples of the maturity of the XMPP technology. There are both commercial and open-source solutions for chat and IM available with many features supporting users and administrators. More interesting however are examples of XMPP clients that are purpose built for the military, providing not only basic features that are useful for military users, but also special enhancements that support military specific chat messaging. The ability to build such

specific capability on an open standard framework is vital to meeting the needs of the various military users, without risking the compromise of interoperability between services or organizational units.

The XML Tactical Chat project studies the application of XMPP based chat/IM in military environments and researches extending XMPP chat to support other command and control functions. Areas of focus include: using XMPP to route DIS-XML messages in support of distributed modeling and simulation, development of an enterprise solution for chat message logging, archiving, querying, and retrieval, and using XMPP to route XML messages drawn from the JC3IEDM to facilitate distributed collaborative operational mission planning. The successful results of these research efforts demonstrate the value that XMPP holds as a technology that can be extended to meet a wide variety of information technology requirements for the military.

The CCSG12 test was significant in that it demonstrated that XMPP chat communications are supported by bandwidth constrained ship borne communications systems meeting or exceeding the needs of existing Navy chat-users. This is a vital experimental result, because for military adoption of XMPP technology to take place, the deployed tools must be able to operate across the full range of tactical military networks (both robust and austere). This test demonstrated that XMPP chat systems are capable of supporting the immediate chat and IM needs of current users from an interface perspective. Finally, this test demonstrated the use of a distributed and federated XMPP server architecture that is well suited for military adoption.

The XTC research with DIS-XML served to demonstrate an example of using XMPP to route XML payload for purposes other than chat and IM. However, the successful importation of this routing mechanism into a modeling and simulation tool, the AUV Workbench, also has provided the means to conduct distributed and collaborative simulations across the internet without the use of multi-cast traffic.

The XTC Chat Logger presents an example of a simply designed and easily implemented solution for enterprise chat message logging, storage, and querying/retrieval. It serves as an exemplar for the benefits of working with XML chat data. The native XML database and XQuery web-application for search provide access to

chat data without the need for special packages for interfacing the database. The use of XMPP client applications demonstrates the flexibility and utility of the available open source XMPP based code sets.

The JC3IEDM-Enhanced Tactical Collaboration experiment within Trident Warrior 2006 demonstrated both the value of an XML expressed data model for military command and control and the value of an XMPP based network for routing and delivery of the XML messages. This experiment specifically exhibits the potential application of XMPP technologies to the Command and Control realm of the DoD.

The need for a comprehensive solution to military chat and instant messaging is profound. This thesis demonstrates that XMPP is a technology that can meet the requirements of the military for Chat and IM while providing interoperability and extensibility solutions that are not afforded by competing solutions. The broad inclusion of XMPP into DoD Command and Control and replacement of inferior, stovepiped, and incompatible protocols will provide a force multiplier now and into the future.

B. RECOMMENDATIONS FOR FUTURE WORK

1. Military Implementation

While the results of testing XMPP chat on naval tactical networks are promising, further testing with XMPP on austere networks is required. Further research and testing using XMPP in very low bandwidth and intermittent connection environments is necessary. Though creating and coordinating opportunities to test new technologies on operational systems is challenging, these efforts are vital to ensuring that future solutions to Command and Control needs are as comprehensive and scalable as possible.

2. Client Development

The JFCOM Transverse project is an excellent model for military development of XMPP client applications. There are military service and organizational requirements that can be met or enhanced with custom client development, and proper implementation of these features would allow for retention of interoperability and future functionality. Additionally, research and development of XMPP chat-client support for handheld and other portable devices are likely to become a requirement as well.

3. JC3IEDM / XMPP Development

The JTC concept is a significant demonstration of the potential capability of XMPP and XML messages. JTC exposed a very small subset of the JC3IEDM data model, and further expansion of this data model and the use of XMPP will lead to significant enhancement of C2 capabilities. JC3IEDM adoption as a data foundation for C2 systems is becoming more prevalent. XMPP is a natural fit into any C2 system that relies of XML messaging. Further development of JTC and other C2 systems that merge JC3IEDM and XMPP is required.

4. XMPP End-to-End Encryption

XMPP has inherent security specifications in its core standard. However, end to end encryption of XMPP payload provides both greater security and additional functionality. Much of the research associated with chat and IM is in relation to joint or coalition environments. Additionally, there is a general need for object encryption and signature with military application of XMPP solutions. End-to-end XML encryption and signature of XMPP message payload will enhance information assurance and security and may provide the basis for multi-level secure chat communications in support of joint and coalition operations.

5. Efficient XML Interchange (EXI)

Though the W3C Working Group for Efficient XML Interchange doesn't adjourn until the end of 2007, this is a current critical area of future research for military application of XMPP. In order for XMPP chat to become a viable solution at all levels of the military, it is likely that a bandwidth and memory efficient implementation will be required. There is needed research into applying XML binary serialization techniques to XMPP streams. XMPP serialization effects on bandwidth efficiency, XMPP routing performance, and message parsing and handling performance are all measures that require future study. Any implementation efforts into applying binary serialization will require namespace awareness as an integral feature.

6. DIS-XML

The IEEE Distributed Interactive Simulation (DIS) specification describes a binary format for Protocol Data Unit (PDU) packets. The DIS-XML approach had been shown to be a worthy alternative encoding that enables the many benefits of XML to be

applied to an important legacy data format. Specification work is now appropriate to rigorously document DIS-XML, thus aligning the IEEE DIS standard with the XML family of standards and the XMPP standard. Some further work is needed to ensure consistent binding of DIS-XML PDUs as XMPP message payloads using proper XML namespace labeling of elements (McGregor et al., 2006).

7. X3D Graphics

The X3D graphics specification (Web3D, 2006) already supports native binary encoding and multicast transport of IEEE DIS PDUs. This support enables active multiplayer participation in shared X3D worlds (Brutzman, 2003). However, such support is ordinarily restricted to local-area networks (LANs) since multicast is usually forbidden to cross firewalls. Adding the DIS-XML encoding and XMPP transport to the X3D graphics specification can enable creation of large-scale virtual environments (LSVEs) across the Internet.

8. Chat Log Search

Search is a powerful capability. More work on XQuery usage in conjunction with XMPP is definitely worthwhile. Comparing XQuery capabilities with the Lucene search engine (Gospodnetic, Hatcher, 2005) is also of value to fully document the potential capabilities of chat search.

9. Chat Log Comparison and Correlation

IN the military domain, communications are often subject to delays or dropouts. Chat servers can overcome many failure modes by keeping recent history available and resynchronizing remote server or clients upon reconnection. However, if this recovery mode is not possible, significant differences in common tactical picture can result. It is worthwhile to examine how maintaining independent chat loggers at remote sites can be used to compare, correlate, and merge chat traffic across a widely distributed set of users. This capability will improve network troubleshooting and (more importantly) allow determination of “what participants received which information when.”

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX A. USER GUIDE FOR THE XTC CHAT LOGGER

The XTC Chat Logger has three functional components, the `ChatMessageLogger.java`, an eXist native XML database, and `xtclog.xql`.

A. CHATMESSAGELOGGER.JAVA

The `ChatMessageLogger.java` relies on a Java properties file, `xmpp.properties` to configure connection, authentication, room registration, and XML document writing settings for the client application. This properties file must either reside in the working directory of `ChatMessageLogger.java` or be appropriately referenced in an IDE package.

The file listed below was used to connect to the NPS `surfaris.cs.nps.navy.mil` server. It is configured to connect the client application to this server under the username of "Chat_Logger" with a password of "logger" and to register with the chat rooms supported by the MUC service `conference.surfaris.cs.nps.navy.mil`. It further configures the client to write all XML chat logs to the /logs subdirectory of the working directory, and to not log four chat rooms: auvw, auvwtest, load, disxml, and jtc-de. The indicated directory for writing the chat logs must exist prior to running the application, or an I/O Exception terminates the application.

```
# Configuration properties for XMPP Chat Logger. This provides the information to
# connect to an XMPP server and MUC rooms. Obviously, since this includes
# passwords, it should be handled with due care.
#
# @author Adrian D. Armold

# The XMPP server to authenticate to
xmppAuthenticationServer = surfaris.cs.nps.navy.mil

# Username used by the Chat Logger
xmppUserName = Chat_Logger

#The password we use to log in
xmppPassword = logger

#The MUC room service
```

```
multiUserChatService = conference.surfaris.cs.nps.navy.mil

#The directory in which the log files should be saved
savedLogsRootDirectory = \logs

#The list of MUC rooms that should not be logged by the logger. Separate by comma.
doNotLog = auvw,auvwtest,load,disxml,jtc-de
```

These properties are accessed by `ChatMessageLogger.java` via a Properties object, and assigned to object variables as indicated in the code presented below.

```
370         try {
371     Properties applicationProps = new Properties();
372     FileInputStream appStream = new FileInputStream(configDir + "/xmpp.properties");
373             applicationProps.load(appStream);
374             appStream.close();
375 
376             xmppAuthenticationServer =
377                applicationProps.getProperty("xmppAuthenticationServer");
378             xmppUserName = applicationProps.getProperty("xmppUserName");
379             xmppPassword = applicationProps.getProperty("xmppPassword");
380     multiUserChatService = applicationProps.getProperty("multiUserChatService");
381     savedLogsRootDirectory = applicationProps.getProperty("savedLogsRootDirectory");
382     noLoggingRooms = applicationProps.getProperty("doNotLog");
383         } catch(IOException ioe) {
384             System.err.println("IOException: " + ioe.getMessage());
385         }
```

Once variable assignment is complete, the client application logs onto the XMPP server and registers with the support MUC rooms, less the rooms on the “doNotLog” list. The code below performs the server login.

```
158     /**
159      * Establishes a connection to the XMPP server and joins the specified chat room.
160      */
161     public void login() {
162         try {
163             // Authenticate to our local XMPP server
164             connection = new XMPPConnection(xmppAuthenticationServer);
165             connection.login(xmppUserName, xmppPassword);
166         } catch(Exception e) {
167             System.out.println(e + " Login failed");
```

```
168         }
169     }
```

The below listed code performs the MUC registration.

```
171     public void registerMUCRooms() {
172         String mucJid = mucRoom + "@" + multiUserChatService;
173         /*Create a discussion history and set it to receive no history
174           this prevents the re-writing of messages if bounced from the
175           server.
176           */
177         DiscussionHistory dh = new DiscussionHistory();
178         dh.setMaxChars(0);
179 
180         // Java 5.0 generics convention
181         ArrayList<String> doLogRooms = new ArrayList<String>();
182         String [] doNotLogRooms = noLoggingRooms.split(",");
183         for (int i = 0; i < doNotLogRooms.length;i++){
184             doNotLogRooms[i] += "@";
185             doNotLogRooms[i] += multiUserChatService;
186         }
187         try {
188             /* Establish a connection to all MUC rooms
189                unless on the doNotLog list from the
190                properties file.
191              */
192           muc  = new MultiUserChat(connection, mucJid);
193           Collection roomNames = muc.getHostedRooms(connection,multiUserChatService);
194           Iterator itr = roomNames.iterator();
195           HostedRoom hr;
196           String roomJid;
197 
198           while (itr.hasNext()){
199 
200                 hr = (HostedRoom)itr.next();
201                 roomJid = hr.getJid();
202 
203                 boolean addRoom = true;
204 
205                 for (int i = 0;i < doNotLogRooms.length;i++){
206                     if (doNotLogRooms[i].equals(roomJid)){
207                         addRoom = false;
208                         break;
209                     }// end if
210                 }// end for
211                 if (addRoom){
212                     doLogRooms.add(new String(roomJid.toString()));
213                     muc  = new MultiUserChat(connection,roomJid);
```

```
214                     muc.join(xmppUserName,"",dh,2000);
215                 }// end if
216             }// end while
217         }
218         catch(Exception e) {
219             System.out.println(e + " MUC Room registration failed");
220        }
```

Once the application is logged in and registered, it is configured to listen for message packets sent to the registered room.

```
222         // Set up a packet filter to listen for only the things we want
223         AndFilter filter = new AndFilter();
224         OrFilter roomFilter = new OrFilter();
225         PacketFilter messageFilter = new PacketTypeFilter(Message.class);
226         filter.addFilter(messageFilter);
227 
228         for (int i=0; i < doLogRooms.size(); i++){
229             FromContainsFilter fromFilter = new FromContainsFilter
230                     ((String)doLogRooms.get(i));
231             roomFilter.addFilter(fromFilter);
232         }
233 
234         filter.addFilter(roomFilter);
```

Messages that pass the filter are then processed. The message is captured as XML, time stamped if required, and appended to an XML file. The XML files contain all messages for one MUC room for 24 hours. The message handling code is presented below.

```
237       // Next, create a packet listener. We use an anonymous inner class for brevity.
238         PacketListener myListener = new PacketListener() {
239 
240             DelayInformation xDelay = null;
241             public void processPacket(Packet ppacket) {
242                 Message currentMessage = (Message) ppacket;
243 
244                 // To gracefully exit the Message Logger from all Chat rooms
245                 if (currentMessage.getBody().equals("*KickMessageLogger")) {
246                     System.exit(0);
247                 }
248                 /* checks if message has delay timestamp info, if not
249                    it adds the extension and stamps it with the current time*/
250                 if (ppacket.getExtension("x","jabber:x:delay") == null){
251                     xDelay = new DelayInformation(new Date());
252                     String from = ppacket.getFrom();
253                     xDelay.setFrom(from);
```

```
254                     ppacket.addExtension(xDelay);
255                 }// end if
256 
257                 String xmlLogString = ppacket.toXML();
258                 xmlLogString += "</Log>";
259                 int index = xmlLogString.lastIndexOf("stamp=\"");
260                 String timeStamp = xmlLogString.substring( index + 7, index + 15);
261 
262                 index = xmlLogString.indexOf("from=\"");
263                 int index2 = xmlLogString.indexOf("\"",index + 6);
264 
265                 String roomJid = xmlLogString.substring(index + 6, index2);
266 
267                 if (roomJid.contains("/")){
268                     index = roomJid.indexOf("/");
269                     roomJid = roomJid.substring(0,index);
270                 }
271                 String filename = savedLogsRootDirectory + File.separator + roomJid;
272                 filename += "." + timeStamp;
273                 filename += ".xml";
274 
275                 if ((new File(filename)).exists()) {
276                     appendMessage(filename,xmlLogString);
277                 } else {
278                     createLogFile(filename);
279                     appendMessage(filename,xmlLogString);
280                 }//end else
281             } // end process packet
282         };// end PacketListener anonymous class
283 
284         connection.addPacketListener(myListener,filter);
```

The chat log files are well-formed XML documents such as the one presented in Figure 58.

```
<?xml version="1.0" encoding="UTF-8"?>
<Log doc="savage@conference.surfaris.cs.nps.navy.mil.20060807.xml">
        <message id="jcl_21" to="chat_logger@surfaris.cs.nps.navy.mil/Smack"
from="savage@conference.surfaris.cs.nps.navy.mil/adarmold" type="groupchat">
                <body>Here is a test message for the MOVES Open House
Presentation</body>
                <x xmlns="jabber:x:delay" stamp="20060807T19:55:32"
from="savage@conference.surfaris.cs.nps.navy.mil/adarmold"/>
        </message>
        <message id="jcl_22" to="chat_logger@surfaris.cs.nps.navy.mil/Smack"
from="savage@conference.surfaris.cs.nps.navy.mil/adarmold" type="groupchat">
                <body>Here is another message</body>
                <x xmlns="jabber:x:delay" stamp="20060807T19:55:46"
from="savage@conference.surfaris.cs.nps.navy.mil/adarmold"/>
        </message>
        <message id="jcl_23" to="chat_logger@surfaris.cs.nps.navy.mil/Smack"
from="savage@conference.surfaris.cs.nps.navy.mil/adarmold" type="groupchat">
                <body>MOVES Open House Aug 9 2006</body>
                <x xmlns="jabber:x:delay" stamp="20060807T19:56:02"
from="savage@conference.surfaris.cs.nps.navy.mil/adarmold"/>
        </message>
        <message id="jcl_24" to="chat_logger@surfaris.cs.nps.navy.mil/Smack"
from="savage@conference.surfaris.cs.nps.navy.mil/adarmold" type="groupchat">
                <body>another MOVES message</body>
                <x xmlns="jabber:x:delay" stamp="20060807T19:56:10"
from="savage@conference.surfaris.cs.nps.navy.mil/adarmold"/>
        </message>
        <message id="jcl_25" to="chat_logger@surfaris.cs.nps.navy.mil/Smack"
from="savage@conference.surfaris.cs.nps.navy.mil/adarmold" type="groupchat">
                <body>this is yet another MOVES message</body>
                <x xmlns="jabber:x:delay" stamp="20060807T19:56:22"
from="savage@conference.surfaris.cs.nps.navy.mil/adarmold"/>
        </message>
        <message id="jcl_26" to="chat_logger@surfaris.cs.nps.navy.mil/Smack"
from="savage@conference.surfaris.cs.nps.navy.mil/adarmold" type="groupchat">
                <body>test message</body>
                <x xmlns="jabber:x:delay" stamp="20060807T19:56:51"
from="savage@conference.surfaris.cs.nps.navy.mil/adarmold"/>
        </message>
        <message id="jcl_27" to="chat_logger@surfaris.cs.nps.navy.mil/Smack"
from="savage@conference.surfaris.cs.nps.navy.mil/adarmold" type="groupchat">
                <body>The Last message for the MOVES Open House Presentation</body>
                <x xmlns="jabber:x:delay" stamp="20060807T19:57:13"
from="savage@conference.surfaris.cs.nps.navy.mil/adarmold"/>
        </message>

        </Log>
```

Figure 58. XTC chat log documents are generated by the `ChatMessageLogger.java` application.

B. EXIST DATABASE

Once the XML chat log files are created, they need to be imported into the eXist database. eXist is an open source native XML database. For XTC, eXist was installed as a webapp within a Tomcat web server. Following the basic installation instructions described at http://exist.sourceforge.net/deployment.html will accomplish this. eXist's installer deploys the database as a web-app inside a Jetty web server container, so, if a web-server is not previously installed on the machine, this is the easiest and recommended installation method.

Once the database is deployed, there are several options for accessing the database and importing documents into it, including a Java admin client, LDAP, XQuery access control, and WebDav. WebDAV is the recommended method for importing the chat logs into the database. Enabling eXist with WebDAV is described at: http://exist.sourceforge.net/webdav.html. Once WebDAV enabled, chat logs can be manually added to the database via drag and drop operations, or the importation of the logs can be automated.

C. XTCLOG.XQL

The last component of the XTC Chat Logger is the XQuery program, `xtclog.xql`. This program performs the web-based presentation, input reception, database query, and response presentation. It serves the same combined purposes of an HTML web page and a JDBC included Java servlet, and does so without the need for translation between HTML, Java, and SQL. `xtclog.xql` is supported by a javascript file and a cascading style sheet file. `xtclog.xql`.

1. xtclog.xql

xtclog.xql is an XQuery program that creates the web browser presentation, database query, and results presentation. The source code is listed below.

```
xquery version "1.0";

declare namespace xtc="http://movesinstitute.org/brutzman/xtc";
declare namespace delay = "jabber:x:delay";
declare namespace event = "jabber:x:event";

declare function xtc:display-page() as element() {
   util:declare-option("exist:serialize", "media-type=text/html method=xhtml"),

   <html>
       <head>
           <title>XTC Chat Log</title>
           <meta http-equiv="Content-Type" content="text/html; charset=UTF-8"/>
           <link type="text/css" href="styles/xtclog.css" rel="stylesheet"/>
           <script language="Javascript" type="text/javascript"
src="scripts/prototype.js"/>
           <script language="Javascript" type="text/javascript"
src="scripts/behaviour.js"/>
           <script language="Javascript" type="text/javascript" src="scripts/xtclog.js"/>
       </head>
       <body>
           <div id="header">
               <h1>XTC Chat Log Database</h1>
           </div>
           <div id="content">
               <div id="query-panel">
                   <input type="text" id="query"/>
                                   <button type="button" id="send-query">Search
Keyword</button>
```

```
                </div>
                <div id="query-result">
                    <a href="#" id="query-close">Close</a>
                    <h3>Query Results</h3>
                    <div id="query-output"/>
                </div>
                <div id="errors"></div>
                <div id="output">
                                    <h3>Log Request Result</h3>
                        <div id="current">
                            <input readonly="readonly" size = "100" type="text"
id="logid"/>
                        </div>
                     <div id="log-output"/>
                </div>
            </div>
        </body>
    </html>
};

declare function xtc:format-number($num as xs:integer) as xs:string {
    if ($num lt 10) then
        concat("0", $num)
    else
        xs:string($num)
};
function xtc:format-time($time as xs:dateTime) as xs:string {
    concat(
        xtc:format-number(hours-from-dateTime($time)), ':',
        xtc:format-number(minutes-from-dateTime($time)), ':',
        xtc:format-number(seconds-from-dateTime($time))
    )
};
function xtc:format-date($time as xs:dateTime) as xs:string{

        concat(
                xtc:format-number(year-from-dateTime($time)),'-',
                xtc:format-number(month-from-dateTime($time)),'-',
                xtc:format-number(day-from-dateTime($time))
        )
};function xtc:display-event($event as element()) as element() {
if ($event instance of element(message)) then

                    let $dateString := xs:string($event/delay:x/data(@stamp))
            let $dateStringFormatted := string-
join((substring($dateString,1,4),substring($dateString, 5,2),substring
($dateString,7)),"-")
            let $messageDate := xs:dateTime($dateStringFormatted )

    return

        <tr>

            <td class="time"><span
class="date">{substring($dateStringFormatted,1,10)}</span> |
                                                                    <span
class="time">{substring($dateStringFormatted,12)}</span></td>
            <td class="nick">{xs:string(substring-after($event/@from,"/") )}</td>
            <td class="message">
              {
                            let $cb := util:function("xtc:highlight", 3)
                return
                    text:highlight-matches($event/body/text(), $cb, ())
            }
            </td>
        </tr>
    else
        ()
};
function xtc:display-date($date as xs:date, $doc as xs:string?) as element()* {
    util:declare-option("exist:serialize", "media-type=text/xml omit-xml-declaration=no"),
```

```
   let $log :=
       if ($doc ) then
           //Log[data(@doc) = $doc ]
       else
           //Log/message/body[@stamp = $date]
   return
       <table>
       {
           for $event in $log/*
           return
               xtc:display-event($event)
       }
       </table>
};

declare function xtc:display-logdoc( $doc as xs:string?) as element()* {
   util:declare-option("exist:serialize", "media-type=text/xml omit-xml-declaration=no"),
   let $log := //Log[data(@doc) = $doc]
   return
       <table>
       {
          for $event in $log/*
           return
               xtc:display-event($event)
       }
       </table>
};

declare function xtc:highlight($term as xs:string, $node as text(), $args as item()+) as
element() {
   <span class="hi">{$term}</span>
};

declare function xtc:query($query as xs:string) as element() {
   util:declare-option("exist:serialize", "media-type=text/xml omit-xml-declaration=no"),
   let $cb := util:function('xtc:highlight', 3)
   let $hits := //message[body &= (concat('"',$query,'"'))]
   return
       <table>
       {
           for $event in $hits
           let $dateString := xs:string($event/delay:x/data(@stamp))
           let $dateStringFormatted := string-
join((substring($dateString,1,4),substring($dateString, 5,2),substring
($dateString,7)),"-")
           let $messageDate := xs:dateTime($dateStringFormatted )
           let $logname := $event/../data(@doc)
           return
               <tr>
                   <td class="time"><span
class="date">{substring($dateStringFormatted,1,10)}</span> |
                       <span
class="time">{substring($dateStringFormatted,12)}</span></td>
                   <td class="nick">{xs:string(substring-after($event/@from,"/") )}</td>
                   <td class="message">
                       <a href="#" onclick="showLogResult('{$logname}')">
                           {text:kwic-display($event/body/text(), 80, $cb, ())}
                       </a>
                   </td>
                   <td class="room">{substring-before($event/../data(@doc),".2006")}</td>
               </tr>
       }
       </table>
};

let $date := request:request-parameter("date", ())
let $query := request:request-parameter("query", ())
let $logdoc := request:request-parameter("logdoc",())
let $log := util:log("DEBUG", $query)
return
```

```
        if ($logdoc) then
                xtc:display-logdoc($logdoc )
        else if ($date) then
            xtc:display-date($date, $query)
    else if ($query) then
        xtc:query($query)
    else

        xtc:display-page()
```

2. xtclog.css

xtclog.css is the accompanying cascading style sheet for xtclog.xql. The code for xtclog.css is listed below.

```
body {
        font-family: Verdana, Arial, Helvetica, sans-serif;
        overflow-y: scroll;
}
* {
    margin: 0;
    padding: 0;
}
#header {
    padding: 10px 0 5px 10px;
    border-bottom: 1px solid blue;
    font-size: 75%;
}
#header h1 {
        font-weight: normal;
        font-size: 150%;
}

#header #menu {
    margin: 0.3em 0 0 0;
    padding: 0;
    float: right;
}

#menu li {
    list-style: none;
    float: left;
    margin: 0;
    white-space: nowrap;
    padding: 0px 20px 0px 20px;
}

#menu li a {
    color: #666666;
}

#content {
    margin: 5px 100px 0 10px;
    font-size: 75%;
}

#query-result {
        margin-top: 5px;
        background-color: #FEFFAF;
        height: 160px;
}

#query-result h3 {
        font-weight: normal;
        font-size: 100%;
```

```
}

#query-close {
	float: right;
}

#query-output {
	background-color: white;
	margin: 5px 3px;
	height: 136px;
	overflow: auto;
}

#query-output td {
	font-size: 70%;
}

#query-output .date {
	color: #cc0099;
}

#query-output .time {
	color: #440099;
}

#query-output .room {
	color: #00ddaa;
}

.hi {
	background-color: yellow;
}

#errors {
	height: 1.5em;
	margin-bottom: 3px;
	color: #AF0024;
}

h2 {
	font-size: 100%;
}

#output {
	background-color: #FC6;
	margin-top: 5px;
	padding: 0 3px 3px 5px;
	border: 1px solid #FFD100;
}

#log-output .time {
	color: #440099;
}

#log-output .time {
	padding-right: 650 px;
}

#log-output .date {
	color: #cc0099;
}

#log-output {
	margin-top: 10px;
	background-color: white;
	overflow: auto;
}


td {
	vertical-align: top;
```

```
        font-size: 80%;
        padding-bottom: 5px;
}

.time {
        padding-right: 10px;
}

.nick {
        padding-right: 25px;
}

.message {
        padding-right: 50px;
}

.action {
        font-style: italic;
}

#navbar {
        width: auto;
        border: 0;
        font-size: 75%;
        margin-bottom: 10px;
}



#current {
        width: 500 px;
        text-align: center;
        font-weight: bold;
        font-size: 100%;
        overflow: auto;
}

#logid {

        border: 0;
        background-color: transparent;
        color: black;
        font-weight: bold;
        text-align: center;
        padding-right: 8px;
}
```

3. xtclog.js

xtclog.js is an accompanying javascript file for xtclog.xql. The code for xtclog.js is listed below.

```
// Call init() after the document was loaded
window.onload = init;
window.onresize = resize;

// Register event handlers for various elements on the page.
// The behaviour library allows to register handlers via CSS
// selectors. This way, we can keep the HTML Javascript-clean.
var behaviourRules = {
        '#next' : function (element) {
                        element.onclick = browseNext;
        },
        '#previous' : function (element) {
                        element.onclick = browsePrevious;
        },
        '#send-query' : function (element) {
                        element.onclick = query;
```

```
        },
        '#query-close' : function (element) {
                        element.onclick = function () {
                                Element.hide('query-result');
                                resize();
                        }
        },
        '#current-date' : function (element) {
                        element.onchange = function () {
                                displayLog(this.value);
                        }
        },
    '#refresh' : function (element) {
            element.onchange = function() {
                var option = this.options[this.selectedIndex].value;
                if (timer) clearInterval(timer);
                if (option != 'off') {
                    refreshPeriod = option * 60 * 1000;
                    timer = setInterval('autoRefresh()', refreshPeriod);
                }
            }
    }
};
Behaviour.register(behaviourRules);

var colors = [
        '#9900cc',
        '#cc0099',
        '#cc9900',
        '#0099cc',
        '#00cc99',
        '#6666ff',
        '#339966',
        '#993366',
        '#669933',
        '#0033cc',
        '#ff99ff',
        '#9900cc',
        '#ffdd00'
]

var nickNames = new Object();
var lastColor = 0;

var currentDate;
var timer = null;

// refresh display every 60 seconds by default
var refreshPeriod = 60 * 1000;

/** onLoad handler to initialize display */
function init() {
        Element.hide('query-result');
        Element.hide('output')
    /*Calendar.setup(
    {
        inputField : 'current-date',
        ifFormat : '%Y-%m-%d',
        button : 'set-date',
        onUpdate : function (calendar) {
                currentDate = calendar.date;
        }
    }
    );*/
    resize();

    Behaviour.apply(); // we need to call behaviour again after this handler
    //displayLog(new Date());
}

/** Resize the query result output div. We want this to have a fixed height,
```

```
 *  so it neatly fits into the browser window.
 */
function resize() {
       var output = $('log-output');
    output.style.height = (document.body.clientHeight - output.offsetTop - 20) + "px";
}

/*function displayLog(date, query) {
       function formatDayMonth(value) {
              if (value < 10)
                     return '0' + value;
              else
                     return value;
       }

       if (timer)
              clearInterval(timer);

       var dateStr;
       if (typeof date == 'string')
              dateStr = date;
       else
              dateStr = date.getFullYear() + '-' + formatDayMonth(date.getMonth() + 1) +
                     '-' + formatDayMonth(date.getDate());
       var params = 'date=' + dateStr;
       if (query)
              params += '&query=' + query;

       var ajax = new Ajax.Request("xtclog.xql", {
                     method: 'post', parameters: params,
                     onComplete: displayResponse,
                     onFailure: requestFailed
              });
       $('errors').innerHTML = 'Retrieving log ...';
       $('current-date').value = dateStr;
       currentDate = date;

       timer = setInterval('autoRefresh()', refreshPeriod);
}*/

function displayLogDoc(logname) {

       if (logname)
              params = '&logdoc=' + logname;

       var ajax = new Ajax.Request("xtclog.xql", {
                     method: 'post', parameters: params,
                     onComplete: displayResponse,
                     onFailure: requestFailed
              });
       $('errors').innerHTML = 'Retrieving log ...';
       $('logid').value = logname;
       //timer = setInterval('autoRefresh()', refreshPeriod);
}

function displayResponse(request)  {
       $('errors').innerHTML = '';
       var output = $('log-output');
       output.innerHTML = request.responseText;
       colorify(output);
       var spans = output.getElementsByTagName('span');
       if (spans.length > 0)
              spans[0].scrollIntoView();
       else {
              var rows = output.getElementsByTagName('tr');
              rows[rows.length - 1].scrollIntoView();
       }

       Element.show('output');
}
```

```
function autoRefresh() {
	$('errors').innerHTML = 'Refreshing ...';
	displayLog(currentDate);
}

function requestFailed(request) {
	$('log-output').innertHTML =
		"The request to the server failed.";
}

function browseNext() {
	var newMillis = currentDate.getTime() + (1000 * 60 * 60 * 24);
	displayLog(new Date(newMillis));
}

function browsePrevious() {
	var newMillis = currentDate.getTime() - (1000 * 60 * 60 * 24);
	displayLog(new Date(newMillis));
}

function query() {
	var qu = $F('query');
	if (!qu || qu.length == 0) {
		alert('Please enter a string to search for!');
		return;
	}

	var params = 'query=' + escape(qu);
	var ajax = new Ajax.Request("xtclog.xql", {
			method: 'post', parameters: params,
			onComplete: queryResponse,
			onFailure: requestFailed
		});
	$('errors').innerHTML = 'Query sent ...';
	Element.show('query-result');
	resize();
}

function queryResponse(request) {
	$('query-output').innerHTML = request.responseText;
	colorify($('query-output'));
	$('errors').innerHTML = '';
}

function showQueryResult(dateStr, query) {
	displayLog(dateStr, query);
}

function showLogResult(log) {
	displayLogDoc(log);
}

function browseToDate(calendar) {
	alert(calendar.date.toString());
}

function colorify(element) {
	var rows = element.getElementsByTagName('tr');
	for (var i = 0; i < rows.length; i++) {
		var columns = rows[i].getElementsByTagName('td');
		if (columns.length == 4 | columns.length == 3) {
			var nick = getElementValue(columns[1]);
			var color = pickColor(nick);
			columns[1].style.color = color;
		}
	}
}
function pickColor(nick) {
	var last = nickNames[nick];
	if (last)
		return last;
```

```
        if (lastColor > colors.length)
                last = 'black';
        else
                last = colors[lastColor++];
        nickNames[nick] = last;
        return last;
}
function getElementValue(node) {
        var val = '';
        var child = node.firstChild;
        while (child) {
                val += child.nodeValue;
                child = child.nextSibling;
        }
        return val;
}
```

xtclog.xql is deployed as a web-application, as eXist resides as a web-app in the Tomcat container. Run the search application through a web browser. http://localhost:8080/exist/xtc/xtclog.xql is the deployed URL used to support this research, but this will be dependent on the installation configuration of the database and XQuery application.

APPENDIX B. COMCARSTKGRU 12 CHAT TEST RESULTS MESSAGE

Following the chat test in October of 2005, the Commander Carrier Strike Group 12, released the, below listed, message. The message details the exercise purpose, design, and results. Recommendations for further testing and adoption of XMPP chat systems are provided as well.

ROUTINE

R 262139Z OCT 05

FM COMCARSTRKGRU TWELVE

TO CDR USJFCOM NORFOLK VA//J6/J9//
USPACOM HONOLULU HI//J6//
HQ SACT//CIS//
AFC2ISRC LANGLEY AFB VA//A6//
DISA WASHINGTON DC//J6//
COMSECONDFLT
COMNAVNETWARCOM NORFOLK VA//N6/N3/N8//

INFO DON CIO WASHINGTON DC
CNO WASHINGTON DC//N71/N6T//
CG MARCORSYSCOM QUANTICO VA
COMFLTFORCOM NORFOLK VA
COMPACFLT PEARL HARBOR HI
COMSPAWARSYSCOM SAN DIEGO CA
COMNAVSEASYSCOM WASHINGTON DC
COMNAVAIRSYSCOM PATUXENT RIVER MD
COMNAVSURFOR SAN DIEGO CA
COMNAVAIRLANT NORFOLK VA
COMNAVSURFLANT NORFOLK VA
COMNAVAIRFOR SAN DIEGO CA
COMTHIRDFLT
COMFIFTHFLT
COMSIXTHFLT
COMSEVENTHFLT
COMPHIBGRU ONE
COMPHIBGRU TWO
COMCARSTRKGRU TWO
COMCARSTRKGRU THREE
COMCARSTRKGRU FIVE
COMCARSTRKGRU SIX
COMCARSTRKGRU SEVEN
COMCARSTRKGRU EIGHT
COMCARSTRKGRU TEN
COMCARSTRKGRU ELEVEN
COMCARSTRKGRU TWELVE

COMEXSTRIKGRU ONE
COMEXSTRIKGRU THREE
COMEXSTRIKGRU FIVE
COMDESRON TWO
NVPGSCOL MONTERY CA
FLENUMMETOCCEN MONTEREY CA
COMDESRON TWO
USS ENTERPRISE
USS ANZIO

UNCLAS //N02325//

MSGID/GENADMIN/COMCARSTRKGRU TWELVE/-/OCT//

SUBJ/RESULTS OF COMCARSTRKGRU TWELVE COMBINED FLEET, JOINT, COALITION /TEST OF OPEN STANDARDS CHAT//

REF/A/RMG/COMCARSTRKGRU TWELVE/172210ZOCT2005/NOTAL//

REF/B/RMG/COMCARSTRKGRU TWELVE/171200ZOCT2005/NOTAL//

REF/C/RMG/COMCARSTRKGRU TWELVE/171201ZOCT2005/NOTAL//

REF/D/DOC/COMCARSTRKGRU TWELVE/08JUN2005/NOTAL//

REF/E/DOC/COMCARSTRKGRU TWELVE/19OCT2005/NOTAL//

NARR/REF A IS COMCARSTRKGRU TWELVE, COMCARSTKGRU TEN, COMCARSTRKGRU EIGHT, COMEXSTRIKGRU ONE CONSOLIDATED MSG DOCUMENTING FLEET REQUIREMENT FOR OPEN STANDARDS ARCHITECTURE. REF B IS COMCARSTRKGRU TWELVE, COMCARSTKGRU TEN, COMCARSTRKGRU EIGHT, COMEXSTRIKGRU ONE MSG DOCUMENTING FLEET REQUIREMENT FOR ENTERPRISE SERVICES. REF C IS COMCARSTRKGRU TWELVE, COMCARSTKGRU TEN, COMCARSTRKGRU EIGHT, COMEXSTRIKGRU ONE MSG DETAILING SPECIFIC ENTEPRISE SERVICES TO SUPPORT AFLOAT COMMAND AND CONTROL. REF D IS THE WHITE PAPER PROPOSING A JOINT/ALLIED COALITION TEST OF OPEN STANDARD EXTENSIBLE MESSAGING PRESENCE PROTOCOL CHAT ON NIPRNET, WHICH INCLUDED TEST OBJECTIVES AND METRICS FOR SUCCESS. REF E IS THE XML TACTICAL CHAT TEST PLAN.//
POC/BARRETT/CDR/COMCARSTRKGRU TWELVE/NORVA/TEL:(757) 444-2600 /EMAIL:DANELLE.BARRETT(AT).NAVY.MIL//
GENTEXT/REMARKS/1. AS COMSECONDFLT'S EXECUTIVE AGENT FOR C4I, COMCARSTRKGRU TWELVE INITIATED AND CONDUCTED A SUCCESSFUL TEST OF OPEN STANDARDS COMPLIANT CHAT TOOLS ON 19 OCT 05 ON NIPRNET INVOLVING USJFCOM, USPACOM/COMPACFLT, DISA, AIR FORCE, NATO, SPAWAR, NAVAL POSTGRADUATE SCHOOL (NPS) AND COMCARSTRKGRU TWELVE AFLOAT UNITS (UNDERWAY ABOARD USS ENTERPRISE AND USS ANZIO). THE PURPOSE OF THIS MESSAGE IS TO DOCUMENT THE TEST RESULTS AND TO MAKE RECOMMENDATIONS ABOUT FLEET OPEN STANDARDS TACTICAL CHAT.
2. BACKGROUND:
A. EXTENSIBLE MESSAGING PRESENCE PROTOCOL (XMPP) IS AN OPEN STANDARDS PROTOCOL FOR CHAT, WHERE THE DATA ARE IN EXTENSIBLE MARKUP LANGUAGE (XML) FORMAT. GOVERNMENT AND INDUSTRY SUPPORT OF A STANDARD IS KEY TO CONTINUED DEVELOPMENT AND GROWTH. ON 20 OCT 05, THE COLLABORATION TECHNICAL WORKING GROUP OF THE DEFENSE INFORMATION TECHNOLOGY STANDARDS REGISTRY (DISR) VOTED TO MAKE XMPP A MANDATORY

STANDARD. THIS MAKES XMPP THE ONLY APPROVED INSTANT MESSAGING STANDARD APPROVED BY THE DISR. IN FY05 DISA FUNDED SPAWAR SYSTEMS CENTER SAN DIEGO TO CONDUCT A BANDWIDTH ANALYSIS OF XMPP OVER TACTICAL COMMUNICATIONS. POC FOR THIS STUDY IS PERRY POWELL (POWELLP(AT)NAVY.MIL) ADDITIONALLY, THE INTERNET ENGINEERING TASK FORCE (IETF) FORMALIZED THE CORE XML STREAMING PROTOCOLS AS AN APPROVED INSTANT MESSAGING AND PRESENCE TECHNOLOGY UNDER THE NAME XMPP. MAJOR COMMERCIAL SUPPORTERS/USERS OF XMPP CHAT INCLUDE: HP, JABBER, INC., ORACLE, SUN MICROSYSTEMS, AT&T, EDS, SONY, ANTEPO, APPLE, HITACHI, JIVE, DESKNOW, RHOMBUS, MERAK, TIPIC, CONVERSANT AND IN AUGUST 2005 GOOGLE ANNOUNCED THAT ITS INSTANT MESSAGING CAPABILITY WOULD BE XMPP
COMPLIANT. THERE ARE COMMERCIAL AND OPEN SOURCE CLIENT/SERVER IMPLEMENTATIONS RUNNING ON SOLARIS, WINDOWS, LINUX, HP-UX, MACOS X, PALMOS, WINDOWS CE, SYMBIAN, AND ANY PLATFORM CAPABLE OF RUNNING JAVA STANDARD (J2EE) OF MICRO (J2ME) EDITIONS.
B. THE FLEET REQUIRES AN OPEN STANDARDS BASED, SECURE, BANDWIDTH FRIENDLY TACTICAL CHAT TOOL (REFS (A)-(C) GERMANE) THAT PROVIDES MAXIMUM EFFICIENCY, SECURITY, AND INTEROPERABILITY WITH OTHER GOVERNMENT AGENCIES AND JOINT/ALLIED/COALITION PARTNERS. THE TOOL MUST BE SCALEABLE AND SUPPORT A FEDERATED SERVER ARCHITECTURE WITH PRESENCE OF USERS AND THEIR STATUS IN THE ENTERPRISE COLLABORATIVE SPACE, PROVIDE PERSISTENT AND TEMPORARY AWARENESS, AND BE ABLE TO SYNCHRONOUSLY TEXT MESSAGES OR SEND AN ASYNCHRONOUS MESSAGE WITH ATTACHMENTS. IT MUST ALSO BE ABLE TO OPERATE IN A DISCONNECTED MODE ON A CLOSED NETWORK (I.E., AFLOAT UNIT WITHOUT SATELLITE CONNECTIVITY).
C. CURRENT FLEET TACTICAL CHAT SITUATION. AFLOAT NAVAL UNITS PRIMARILY USE MIRC ON WINDOWS WORKSTATIONS, MS CHAT ON IT21 SHIPS, AND ZIRCON CHAT ON GLOBAL COMMAND AND CONTROL SYSTEM- MARITIME (GCCS-M) AS TACTICAL CHAT TOOLS WITHIN THE FLEET BOUNDARY 1 FIREWALL ON SIPRNET. CHAT IS NOT USED ON NIPRNET. SAMETIME MEETING/CHAT IS IN LIMITED USE ON SIPRNET AND CENTRIXS WITHIN THE FLEET AND WITH OTHER EXTERNAL UNITS. SOME UNITS ALSO USE MULTI-LEVEL SECURE CHAT, A GOVERMENT DEVELOPED SOFTWARE BASED ON THE DABBLE PROTOCOL (NON-OPEN STANDARD CODE). THE TOOL PRIMARILY USED FOR DAY-TO-DAY TACTICAL CHAT IS IRC CHAT. IRC HAS INHERENT SECURITY VULNERABILITIES AND LITTLE ACTIVE COMMERCIAL DEVELOPMENT. THE DEFENSE COLLABOARTIVE TOOL SUITE (DCTS) IS AN INTEGRATED SET OF OFF THE SHELF APPLICATIONS FOR COLLABOARTION. WHILE DCTS TOOLS CONFORM TO OPEN STANDARDS FOR VIDEO AND TEXT CHAT, THE APPLICATIONS CHOSEN BY DCTS GENERALLY DO NOT IMPLEMENT STRONG SECURITY MECHANISMS INCLUDING COMMUNICATIONS. ADDITIONALLY, DCTS IS NOT BANDWIDTH FRIENDLY SO IS NOT USED BY NAVAL UNITS AFLOAT. SOME NAVAL FLAG SHIPS ALSO USE INFO WORKSPACE (IWS) WHICH REQUIRES A LARGE AMOUNT OF BANDWIDTH, USES AN EXPENSIVE NON-OPEN STANDARDS CLIENT, AND FEDERATION OF IWS SERVERS IS DIFFICULT IN A BANDWIDTH DISADVANTAGED ENVIRONMENT.
D. USE OF AN XML BASED CHAT SOLUTION WILL ALLOW NAVY TO LEVERAGE XML DATA GUARDS (SUCH AS THE USJFCOM XML DATA GUARD CURRENTLY IN TESTING WITH THE NATIONAL SECURITY AGENCY) FOR CROSS DOMAIN CHAT. THIS WILL ENABLE MULTI-USE OF A SINGLE GUARD TOOL FOR XML RELATIONAL DATABASES, XML CHAT AND XHTML WEB DATA WILL IMPROVE INTEROPERABILITY WITH OTHER OPEN STANDARDS PRODUCTS AND WILL ELIMINATE THE NEED FOR PROPRIETARY CROSS DOMAIN TOOLS CURRENTLY IN PLACE.
3. TEST OBJECTIVES AS IDENTIFIED IN REFS (D) AND (E) WERE TO:
A. CONNECT JIVE MESSENGER AND JABBER XCP 4.2.3 SERVERS AT NAVAL

POSTGRADUATE SCHOOL AND JFCOM RESPECTIVELY. ENSURE PRESENCE OF USERS AND PERSISTENCE BETWEEN USERS ON BOTH SERVERS.
B. LOAD AND TEST DIFFERENT XMPP COMPLIANT CHAT CLIENTS AT SEVERAL JOINT AND COALITION COMMANDS, INCLUDING NAVY UNITS AT SEA. THESE MUST INCLUDE BOTH THICK AND WEB BASED CLIENTS. INTEROPERABILITY AMONG THE DIFFERENT CLIENTS MUST BE VERIFIED.
C. HOLD CHAT SESSION WITH ALL PARTICIPANTS. MONITOR BANDWIDTH UTILIZATION OF AFLOAT CONNECTIONS AND OTHER LOCATIONS WHERE DATA COULD BE COLLECTED. ANALYZE BANDWIDTH DATA TO DETERMINE FUNCTIONALITY OF CLIENTS IN A BANDWIDTH DISADVANTAGED ENVIRONMENT (SHIPS AT SEA).
D. COLLECT SUBJECTIVE DATA FROM USERS ON THE FUNCTIONALITY AND PERFORMANCE OF THE DIFFERENT XMPP COMPLIANT CLIENTS.
4. TEST ARCHITECTURE:
A. SERVERS.
(1) JFCOM SERVER: RUNNING ON A DELL 2650, DUAL 3.0GHZ CPUS, FOUR GIG OF MEMORY. SOFTWARE INCLUDED RED HAT ENTERPRISE LINUX AS 3.0 AND JABBER XCP 4.2.5 OPERATING IN THE UNCLASSIFIED DEFENSE RESEARCH AND ENGINEERING NETWORK (DREN), SPEED 0C-48.
(2) NAVAL POSTGRADUATE SCHOOL: RUNNING ON AN INTEL DUAL PROCESSOR P3 750 MHZ WITH ONE GB OF MEMORY AND 40 GB OF DISK SPACE. SOFTWARE OF OPERATING SYSTEM WAS FEDORA CORE 3 LINUX. THE XMPP SERVER WAS JIVE MESSENGER 2.3.0 AND THE CONNECTION OF THE SERVER TO THE NETWORK WAS VIA THE DREN. SERVER LOAD WAS MINIMAL DURING THE TEST.
(3) THE SERVERS WERE CONNECTED USING XMPP'S STANDARD SERVER TO SERVER ENCRYPTED COMMUNICATIONS ON TCP PORT 5269.
B. CLIENTS. CLIENTS AT THE NAVAL POSTGRADUATE SCHOOL CONNECTED TO THE SERVER IN MONTEREY. ALL OTHER CLIENTS CONNECTED TO THE JFCOM SERVER.
USJFCOM: BUDDYSPACE VERSION 2.5.1 PRO WITH J9 ENHANCEMENTS
USPACOM/COMPACFLT: BUDDYSPACE VERSION 2.5.1 PRO WITH J9 ENHANCEMENTS
NATO: BUDDYSPACE VERSION 2.5.1 PRO WITH J9 ENHANCEMENTS AND JABBER SSL WEB CLIENT IN NON-POLLING MODE.
AIR FORCE: JABBER WEB CLIENT IN POLLING MODE OVER PORTS 80 AND 443, JABBER WEB SSL CLIENT IN NON-POLLING MODE OVER PORTS 5222 OR 5223 AND BUDDYSPACE VERSION 2.5.1 PRO WITH J9 ENHANCEMENTS
NAVAL POSTGRADUATE SCHOOL: EXODOUS THICK CLIENT VERSION 0.9.1 ON WINDOWS XP, AND VIA ICHAT 3.0.1 ON MAC OS 10.4.
SPAWAR: JABBER MESSENGER 3.0.2.2 THICK CLIENT, JABBER SSL WEB CLIENT AND JABBER WEB SSL CLIENT.
USS ENTERPRISE: BUDDYSPACE VERSION 2.5.1 PRO WITH J9 ENHANCEMENTS
USS ANZIO: BUDDYSPACE VERSION 2.5.1 PRO WITH J9 ENHANCEMENTS
5. LESSONS LEARNED:
A. BANDWIDTH ANALYSIS: OVER A ONE AND A HALF HOUR TEST PERIOD, SERVER BANDWIDTH MONITORING CAPTURED 10 MB OF CLIENT-SERVER DATA COMMUNICATIONS FOR CHAT AND INSTANT MESSAGING. TEST PARTICIPANTS RECEIVED UP TO 600 KB OF TCP MESSAGE COMMUNICATIONS FROM THE SERVER. THE MOST ACTIVE USERS SENT UP TO 100 KB OF TCP MESSAGE COMMUNICATIONS TO THE SERVER. THE DATA AMOUNT VARIED WITH THE TIME USERS ENTERED AND THE AMOUNT OF ONE-TO-ONE MESSAGES. FROM THESE DATA, IT IS ESTIMATED THAT PASSIVE USERS AVERAGED 0.11 KB PER SECOND AND ACTIVE USERS AVERAGED 0.13 KB PER SECOND.
B. PERFORMANCE AND HUMAN MACHINE INTERFACE (HMI):
FOUR USERS UNDERWAY WERE INVOLVED WITH THE TEST (TWO ON ANZIO AND TWO ON ENTERPRISE). IT WAS IMPORTANT TO TEST ON A LARGE DECK SHIP WHICH HAS MORE BANDWIDTH AND REDUNDANT SATELLITE LINKS AND ON A

SMALLER, MORE BANDWIDTH DISADVANTAGED SHIP. OTHER NETWORK USER ACTIVITY WAS NOT RESTRICTED AND BANDWIDTH WAS NOT INCREASED OR MODIFIED TO MAINTAIN CONSISTENCY WITH REAL-WORLD OPERATIONS. ON ENTERPRISE WITH HUNDREDS OF PERSONNEL ON LINE AND WITH ONLY 786 KBPS BANDWIDTH ON NIPRNET, THE CHAT (BUDDYSPACE THICK CLIENT) USED PERFORMED AS GOOD OR BETTER THAN EXISTING CLIENTS, BASED ON FEEDBACK FROM THOSE TESTING. THE SAME WAS TRUE ON USS ANZIO WHERE THE BANDWIDTH ON NIPRNET WAS 256 KBPS (WITH LESS ACTIVE USERS). IN THE AFTER TEST SURVEY, ALL AFLOAT USERS RATED IT A FIVE ON A SCALE OF ONE TO FIVE WITH ONE BEING THE LOWEST. OVER A ONE AND A HALF HOUR TEST PERIOD, USERS MAINTAINED THEIR CONNECTION THE ENTIRE TIME ON USS ANZIO AND DROPPED CONNECTION FOUR TIMES ON ENTERPRISE. (THIS COULD ALSO HAVE BEEN ATTRIBUTED TO AN INTERNAL NIPRNET LAN CASUALTY ON ENTERPRISE THAT HAD DEGRADED LARGE PORTIONS OF THE NETWORK). AS THE CHAT ENTRIES IN BUDDYSPACE WERE TIME STAMPED AND PERSISTENT, RE-ENTERING THE CHAT ROOM POSED NO LOSS OF SITUATIONAL AWARENESS. IT WAS INITIALLY PLANNED TO TEST THE WEB-BASED JAVA CLIENT ON USS ENTERPRISE AS WELL BUT PRE-TESTING PIERSIDE CONCLUDED THAT THE SLOW RESPONSE OF THE WEB CLIENT MADE IT INEFFECTIVE. THIS WAS THE CASE EVEN WITH THE OTHER SHORE BASED UNITS USING THE WEB CLIENT. AIR FORCE REPORTED EXTREMELY POOR PERFORMANCE OF THE WEB CLIENT AND RECOMMENDED THAT IT MUST MATURE AND IMPROVE BEFORE BEING USED OPERATIONALY. BASED ON RESULTS OBSERVED WITH BOTH OPTIONS, A THICK CLIENT CHAT SOLUTION REMAINS THE BEST ALTERNTIVE FOR AFLOAT UNITS, PARTICULARLY THOSE WITH LIMITED BANDWIDTH.

C. FEDERATION OF SERVERS: AS NAVY WILL OPERATE ANY CHAT ARCHITECTURE IN A FEDERATED MANNER, IT WAS IMPORTANT TO DEMONSTRATE PRESENCE OF USERS AND THEIR STATUS DURING THE ENTIRE DEMONSTRATON. TWO SERVERS WERE FEDERATED TOGETHER VIA A SERVER TO SERVER CONNECTION.

D. SECURITY: THE AUTHENTICATION, TIME STAMP, AND PERSISTENT SESSION FEATURES IN THE BUDDYSPACE CLIENT WERE USEFUL FROM AN INFORMATION ASSURANCE PERSPECTIVE.

E. PLANNING FOR THE DEMONSTRATION BEGAN IN JUNE 2005, AS DID SUBMISSION OF PAPERWORK NECESSARY TO LOAD THE CHAT CLIENT SOFTWARE AFLOAT. THE LABYRINTH OF REQUIREMENTS, DIFFERENT ORGANIZATIONS, AND APPROVALS TO LOAD AFLOAT WERE STAGGERING AND CONSUMED HUNDREDS OF MAN-HOURS. THESE PROCESSES (SHIPMAIN, PPL, IATO/FLEET FIREWALL APPROVAL ETC.) DO NOT INCLUDE MECHANISMS TO SUPPORT NON-PERMANENT INSTALLATIONS FOR FLEET INITIATED BETA TESTING OR TECHNOLOGY DEMONSTRATIONS. FINAL APPROVAL VIA THE SHIPMAIN PROCESS TO LOAD THE CLIENT SHIPBOARD WAS NOT RECEIVED UNTIL THE DAY OF TESTING AND TWO WEEKS BEFORE THE TEST DATE IT WAS DISCOVERED THAT FUNCTIONAL AREA MANAGER APPROVAL SHOULD HAVE BEEN OBTAINED BUT WAS WAIVED. COMCARSTRKGRU TWELVE WILL BE SENDING A SEPARATE MESSAGE TO DISCUSS THESE ISSUES IN MORE DETAIL.

6. RECOMMENDATIONS:

A. NETWARCOM CONSIDER A POLICY MAKING XMPP THE APPROVED OPEN STANDARD CHAT PROTOCOL FOR THE FLEET AND SHORE NAVY ON NIPRNET AND SIPRNET. AUTHORIZE THE PERMANENT CHANGE IN FIREWALL AND PROXY POLICIES TO ALLOW THE USE OF XMPP.

B. FORCENET ARCHITECTS DEVELOP AND SYSCOMS (SPAWAR, NAVSEA, NAVAIR) IMPLEMENT A FEDERATED, XMPP COMPLIANT CHAT SOLUTION FOR THE FLEET AND ELIMINATE NON-XMPP COMPLIANT CHAT PROGRAMS. EACH SHIP SHOULD HAVE ITS OWN XMPP COMPLIANT CHAT SERVER TO BE ABLE TO CONTINE TO OPERATE INTERNALLY DURING PERIODS WHEN DISCONNECTED FROM THE SATELLITE LINK. REPLICATION AND SYNCHRONIZATION OF SERVER DATA

SHOULD BE CAREFULLY ENGINEERED.
C. NETWARCOM WORK WITH SYSCOMS (SPAWAR, NAVSEA AND NAVAIR) TO COLLECTIVELY CONSIDER USING BUDDYSPACE (OPEN STANDARD, OPEN SOURCES GOVERNMENT OFF THE SHELF SOFTWARE DEVELOPED BY JFCOM BASED ON THE JABBER IM MODEL) AS THE SOFTWARE IS FREE AND COULD BE A COST SAVING OVER EXISTING CHAT SOFTWARE IN THE FLEET WITH LICENSING FEES.
D. NETWARCOM LEVERAGE WORK DONE BY NAVAL POSTGRADUATE SCHOOL (DR. DON BRUTZMAN) AND USJFCOM TO APPLY COMPRESSION ALGORITHMS TO XML CHAT, WHICH WILL IMPROVE BANDWIDTH EFFICIENCIES AFLOAT. CURRENT RESEARCH AND TESTING ACHIEVES COMPRESSION OF XML CHAT BY A RATIO OF 3:1, WITHOUT INCREASING LATENCY OF THE CHAT SESSION.
E. NAVY REPRESENTATIVES TO THE DISA NETWORK CENTRIC ENTERPRISE SERVICES WORKING GROUP SUPPORT XMPP COMPLIANT, BANDWIDTH FRIENDLY SOLUTIONS FOR THE FOLLOW ON TO DCTS.
F. CONTINUE TO TEST XMPP, AND OTHER OPEN STANDARDS COMPLIANT COLLABORATIVE TOOLS, IN A JOINT/COALITION AND INTERAGENCY ENVIRONMENT. RECENT EVENTS SUCH AS THE TSUNAMI RELIEF AND HURRICANES KATRINA AND RITA DEMONSTRATED THE REQUIREMENT FOR THIS TYPE OF COLLABORATION VIA UNCLASSIFIED CHANNELS. DO NOT RECOMMEND REQUIRING PUBLIC KEY INFRASTRUCTURE (PKI) CERTIFICATION FOR CLIENTS TO CONNECT AS ALLIED, COALITION, INTERAGENCY, AND NON-GOVERNMENTAL ORGANIZATIONS WOULD BE EXCLUDED FROM COLLABORATION.
G. IAW REFS B AND C, CONSIDER XMPP CHAT AND ALL COLLABORATIVE TOOLS AS ENTERPRISE SERVICES. ENSURE NAVY MARINE CORPS INTRANET (NMCI) ADOPTS XMPP AS ITS INSTANT MESSAGING AND TEXT CHAT SOLUTION AND THAT AN IMPROVED XMPP CLIENT BE INSTALLED ON ALL NMCI WORKSTATIONS. THIS IS PARTICULARLY IMPORTANT FOR EMBARKABLE STAFFS.
H. STREAMLINE PROCESSES TO SUPPORT TEMPORARY INSTALLATIONS AFLOAT FOR CONTROLLED FLEET INITIATED EXPERIMENTATION.
7. COMCARSTRKGRU TWELVE STANDS READY TO ASSIST IN ANY WORKING GROUPS TO FURTHER THIS EFFORT AND FUTURE TECHNOLOGY DEMONSTRATIONS/TESTING OF OPEN STANDARDS CHAT SOLUTIONS AFLOAT.//

BT
NNNN

APPENDIX C. TW06 VIGNETTE PLAYBOOK

The JTC exercise with Trident Warrior 2006 was conducted by eight people over a four day period. During the test period, user/actors were given a series of vignettes, in which they were required to perform a set of planning functions with a specified set of tools. This appendix includes the vignette descriptions provided to the users during the experiment.

TAB 1

Maritime Interdiction using PowerPoint and email

Players:

BHR – DDG1 and SH-60B-1

NPS – DDG2 and SH-60B-2

NUWC – DDG3 and SH-60B-3

Initial Situation:

Reports of ships carrying material that could be used to make WMD have come in causing three DDGs to be deployed off the coast of country X. Further intelligence shows that there are two more shipments of the suspected cargo scheduled to leave the Port Angeles aboard two merchants named “Sneaky Catch” and “Wool Took.” The Port Angeles is a fairly busy port, at 48.2N 123.9W with merchant vessels of all types coming in and out via two traffic lanes. The three DDGs are to conduct Maritime Interdiction Operations in the area outside the Port Angeles, identify the two ships and confiscate the questionable cargo. US ships shall not encroach within 12 miles of the coastline. NUWC is OTC.

PowerPoint/email:

Initial requirements:

PowerPoint slide with chart and shipping lanes created

Tracks of merchants injected into JTC.

All: share slides to show areas of responsibility and use chat for radio comms between DDGs. Tracks will be displayed on JTC. JTC will only be used to display track information.

TAB 2

Maritime Interdiction using JTC and chat

Players:

BHR – DDG1

NPS – DDG2

NUWC – DDG3

Initial Situation:

Reports of ships carrying material that could be used to make WMD have come in causing three DDGs to be deployed off the coast of country X. Further intelligence shows that there are two more shipments of the suspected cargo scheduled to leave the Port Angeles aboard two merchants named “Sneaky Catch” and “Wool Took.” The Port Angeles is a fairly busy port, at 48.2N 123.9W with merchant vessels of all types coming in and out via two traffic lanes. The three DDGs are to conduct Maritime Interdiction Operations in the area outside the Port Angeles, identify the two ships and confiscate the questionable cargo. US ships shall not encroach within 12 miles of the coastline. NUWC is OTC.

JTC:

Initial requirements:

2-3 Opareas to act as merchant shipping lanes

Many tracks for different merchants, two of which are the specific targets.

3 DDGs

Items expected to be generated:

Optasks for areas to be patrolled or searched

Optasks for ships to conduct maritime interdiction

Deliberate Planning:

All: Retrieve OPPLANS “North Shipping Lane” and “South Shipping Lane” from JC3IEDM

All: Each node develop a plan to patrol the given area for the cargo of interest. Begin each OPTASK name with the originating node (Ex: “NUWC MIO zone 1”)

BHR 0000-0800, NPS 0800-1600, NUWC 1600-2400.

NOTE: assigned times are to reduce number of current OPTASKs and clutter on the display and plans generated are completely independent.

Observer at each node type *Report in the observer channel when your node completes planning. Save OPPLAN to JC3IEDM when all nodes have completed planning.

All: Discuss which plan will be used to complete the objective.

Monitoring/Collaborative Planning:

Ships use JTC and chat to assign units to investigate the various merchants coming in and out of port by creating a small task area around the track.

<Scenario ends when both suspect merchants have been found.>

TAB 3

Antisubmarine Warfare using PowerPoint and email

Players:

BHR – DDG1 and ASWC

NPS – DDG2 and SH-60B

NUWC - SSN

Initial Situation:

Intelligence reports indicate that a country X SS may be returning to the Port Angeles from a training exercise to the west and may have hostile intent towards any US forces off its coast. To help with the search for the SS, a 688i class SSN has entered the area and is the best sensor to detect and track the SS. The goal is to establish a track of the SS and determine its intentions prior to engaging. No ship has authorization to fire upon the SS until its intentions have been determined to be hostile.

The ASWC is aboard DDG1 and is the OTC of the area. The DDGs each have the capability to fly SH-60Bs for the purpose of ASW (up to a 150nm range with 1 hour on station). The SSN will remain at periscope depth unless it becomes necessary to go deep to track the SS.

PowerPoint/email:

Initial requirements:

PowerPoint slide with chart and shipping lanes created

Tracks will be displayed on JTC. JTC will only be used to display track information.

Deliberate Planning:

All: Each node independently set up search areas for the three DDGs, their SH-60Bs and the SSN. Then, email your slide to the other two nodes.

Observer at each node type *Report in the observer channel when your node completes planning.

The SSN search area is also its assigned water space and will need to be formally changed in the event that the SSN needs to be relocated to track the SS.

Once complete, discuss the results and establish the ordered plan.

<Inject track of SS>

DDG/SH-60B: detects enemy SS and begins to track. Shortly after gaining the SS, all the SH-60Bs will need to return to their ships due to fuel.

Monitoring/Cooperative Planning:

ASWC: Task the SSN with tracking the SS, collaborate with SSN to reassign water, if needed. Establish an area that if the SS enters, the SSN should check in for further guidance. The area is defined by 46.6N 125.8W

46.6N 127.7W

47.8W 127.7W

47.8N 125.8W

The purpose of this area is to indicate that the SS is showing hostile intent towards the group and needs to be sunk.

SSN: acknowledges the task.

When the SS enters the area:

SSN: check in with ASWC and request authorization to fire upon the SS.

ASWC/DDGs: evaluate need to clear the area of the SS to give SSN a clear shot. Authorize the SSN to shoot.

**Report*

SSN: acknowledge order.

5 min. later:

SSN: reports that the SS is sunk. (Ignore track now)

ASWC: route all assets to new operating areas.

**Report*

TAB 4

Antisubmarine Warfare using JTC and chat

Players:

BHR – DDG1 and ASWC

NPS – DDG2 and SH-60B

NUWC - SSN

Initial Situation:

Intelligence reports indicate that a country X SS may be returning to the Port Angeles from a training exercise to the west and may have hostile intent towards any US forces off its coast. To help with the search for the SS, a 688i class SSN has entered the area and is the best sensor to detect and track the SS. The goal is to establish a track of the SS and determine its intentions prior to engaging. No ship has authorization to fire upon the SS until its intentions have been determined to be hostile.

The ASWC is aboard DDG1 and is the OTC of the area. The DDGs each have the capability to fly SH-60Bs for the purpose of ASW (up to a 150nm range with 1 hour on station). The SSN will remain at periscope depth unless it becomes necessary to go deep to track the SS.

JTC:

Objects/things needed:

Initial requirements

3 DDG

3 SH-60B

1 SSN

1 SS

Track inject of SS

Items expected to be generated:

Operating/Search Areas for DDGs, SH-60Bs, SSN

Area to indicate aggression

New Operating areas for SSN

Deliberate Planning:

All: Each node independently set up search areas for the three DDGs, their SH-60Bs and the SSN Begin each OPTASK name with the originating node (Ex: “NUWC MIO zone 1”)

BHR 0000-0800, NPS 0800-1600, NUWC 1600-2400.

NOTE: assigned times are to reduce number of current OPTASKs and clutter on the display and plans generated are completely independent.

Observer at each node type *Report in the observer channel when your node completes planning. Save OPPLAN to JC3IEDM when all nodes have completed planning.

The SSN search area is also its assigned water space and will need to be formally changed in the event that the SSN needs to be relocated to track the SS.

Once complete, discuss the results and establish the ordered plan.

<Inject track of SS>

DDG/SH-60B: detects enemy SS and begins to track. Shortly after gaining the SS, all the SH-60Bs will need to return to their ships due to fuel.

Monitoring/Cooperative Planning:

ASWC: Task the SSN with tracking the SS, collaborate with SSN to reassign water, if needed. Establish an area that if the SS enters, the SSN should check in for further guidance. The area is defined by 46.6N 125.8W

46.6N 127.7W

47.8W 127.7W

47.8N 125.8W

The purpose of this area is to indicate that the SS is showing hostile intent towards the group and needs to be sunk.

SSN: acknowledges the task.

When the SS enters the area:

SSN: check in with ASWC and request authorization to fire upon the SS.

ASWC/DDGs: evaluate need to clear the area of the SS to give SSN a clear shot. Authorize the SSN to shoot.

**Report*

SSN: acknowledge order.

5 min. later:

SSN: reports that the SS is sunk. (Ignore track now)

ASWC: route all assets to new operating areas.

**Report*

TAB 5

Strike Warfare using PowerPoint and email

Players:

BHR – SSN

NPS – LAC and DDG1

NUWC - TSC and DDG2

Initial Situation:

Tensions are growing with country X. Country X has vowed defend its sovereignty with strikes against the US naval ships off of its coast.

Coastal surveillance and Special Forces units have identified several key targets for tomahawk strikes. The TSC is aboard DDG1, the LAC is aboard DDG2. DDG2 is OTC and the SSN shall remain at periscope depth for the duration of the strike.

NOTE: There is a Unified Force assembled on land with objectives to advance on key locations. However, not all units have established communications.

PowerPoint/email:

Initial requirements:

PowerPoint slide with chart and shipping lanes created

List of friendly forces and neutral sites not to be flown over.

TSC: mark the slide with assigned areas of water for the strike. Send an email with assets (two DDGs and one SSN) and their assigned targets using the INDIGO msg template.

	Target	FPPWP
Target 1	47.4N 122.6W	48.4N 122.9W
Target 2	47.3N 122.4W	47.8N 123.6W
Target 3	47.7N 122.2W	48.3N 124.5W
Target 4	47.0N 122.9W	45.8N 123.9W
Target 5	47.5N 117.7W	48.9N 122.7W
Target 6	45.6N 122.5W	45.0N 124.0W

LAC: mark the slide with safety concerns/no fly areas, etc.

DDG1/DDG2/SSN: send slide overlays via email to the LAC including routes to FPPWP for evaluation.

DDG1, DDG2, and SSN: create intended strike plans for assigned targets.

LAC: evaluate the intended strike plans for safety, give go-ahead, if applicable.

TSC: give hot call.

<DDG2 has a problem with launching one of its assigned missions>

Monitoring/Collaborative Planning:

TSC: reassign the mission to one of the other units.

<Note time>

<Another blue unit checks in and shows up on chart as a friendly track in the area of some of the strikes>

ALL: Discuss possible courses of action for having the friendly unit in the area.

<Note time>

TAB 6

Strike Warfare using JTC and chat

Players:

BHR – SSN

NPS – LAC and DDG1

NUWC - TSC and DDG2

Initial Situation:

Tensions are growing with country X. Country X has vowed defend its sovereignty with strikes against the US naval ships off of its coast.

Coastal surveillance and Special Forces units have identified several key targets for tomahawk strikes. The TSC is aboard DDG1, the LAC is aboard DDG2. DDG2 is OTC and the SSN shall remain at periscope depth for the duration of the strike.

NOTE: There is a Unified Force assembled on land with objectives to advance on key locations. However, not all units have established communications.

JTC:

Objects/things needed:

Initial requirements

List of friendly forces and neutral sites not to be flown over.

1 LHD

1 LPD

1 LSD

3 DDGs

Items expected to be generated:

Optask for the spot of water for each ship to be in during the strike

Optasks for no-fly and separation zones based on the water assignments

Optasks for different units to strike different targets.

Points for target and 1st preplanned waypoint for each target/mission

Tracks for each mission

Deliberate Planning:

TSC: assign areas of water for the strikes. Publish targets and assign to assets (two DDGs and one SSN).

	Target	FPPWP
Target 1	47.4N 122.6W	48.4N 122.9W
Target 2	47.3N 122.4W	47.8N 123.6W
Target 3	47.7N 122.2W	48.3N 124.5W
Target 4	47.0N 122.9W	45.8N 123.9W
Target 5	47.5N 117.7W	48.9N 122.7W
Target 6	45.6N 122.5W	45.0N 124.0W

LAC: Using the assigned water space, publish the safety picture IAW the included LAC intentions message.

**Report*

DDG1, DDG2, and SSN: create intended strike plans for assigned targets.

**Report*

LAC: evaluate the intended strike plans for safety, give go-ahead, if applicable.

TSC: give hot call.

<DDG2 has a problem with launching one of its assigned missions>

Monitoring/Collaborative Planning:

TSC: reassign the mission to one of the other units.

<Another blue unit checks in and shows up on chart as a friendly track in the area of some of the strikes>

ALL: Discuss possible courses of action for having the friendly unit in the area.

**Report*

TAB 7

Amphibious Warfare/Minefield Avoidance using PowerPoint and email

Players:

BHR - MWC/DDG1

NPS - DDG2

NUWC - SSN

Initial situation:

The ESG, consisting of 3 DDGs, 1 LHD, 1 LPD, 1 LSD and 1 SSN, is now to conduct amphibious warfare, landing near port Y in order to support the Unified Force that is already assembled in the area.

The landing area is defined by: 47.00N 123.71W

47.00N 123.89W

46.90N 123.87W

46.90N 123.70W

However, one of the DDGs spotted what could have been mine-laying activity near the intended landing point. There are no other feasible places to land the force in the area though, so the possible minefield needs to be found and plotted in order to make a safe landing.

The suspected mined area roughly covers the area defined by:

46.65N 124.50W

46.65N 124.10W

47.10N 124.20W

47.10N 124.50W

An SSN is deployed with a UUV on board that could investigate the suspected mined region. The goal is to chart any found mines and plan a q-route for surface ships to go through in order to support amphibious operations.

The UUV's mission includes two rendezvous points, one relatively close to the mission area (point A at 46.70N 124.60W) and one that is well clear of the area (point B at 46.2N 124.9W). The idea being that in optimal conditions, the submarine will be able to relay any discovered mines back to the fleet faster by using point A.

Submarine will remain at periscope depth outside of visual range of sortie area and rendezvous points until time to recover UUV.

PowerPoint/email:

Initial requirements:

PowerPoint slide for Loitering Area

PowerPoint slide for Harbor Approach

Deliberate Planning:

All: Create water space plan for all ships in the area to loiter before the amphibious strike

Observer at each node type *Report in the observer channel when your node completes planning. Save OPPLAN to JC3IEDM when all nodes have completed planning.

MWC tells the submarine where to put the UUV mission using Harbor Chart.

SSN acknowledges data.

**Report*

SSN then transmits the data to UUV.

UUV (observer at NUWC) acknowledges data input.

UUV is deployed

SSN reports UUV deployed.

Monitoring/Adaptive Planning:

5 min After the UUV is deployed, a surface ship in the area notices that a large merchant is loitering in the vicinity of rendezvous point A. (Monitor hands this information to DDG2)

DDG2 relays the information to the submarine via radio (chat).

<Inject solution on merchant onto chart/map of having zero speed, close to point A>

SSN orders UUV to rendezvous at point B via ACOMMS.

UUV (observer at NUWC) acknowledges change

**Report*

Rendezvous 3 min later.

After the rendezvous, the data is uploaded to the SSN.

SSN relays the mine data to the fleet via slide, acknowledging the ability for total accuracy is not possible with the given PowerPoint Slide.

Mined Area1:	46.73N 124.10W	Mined Area2:	46.89N 124.18W
	46.77N 124.80W		47.00N 124.26W
	46.89N 124.13W		47.02N 124.18W
	46.90N 124.06W		46.91N 124.13W
Mined Area3:	46.84N 124.23W	Mined Area4:	46.77N 124.22W
	46.89N 124.25W		46.79N 124.27W
	46.87N 124.17W		46.83N 124.19W
Mined Area5:	46.92N 124.23W		
	46.93N 124.31W		
	46.98N 124.28W		

*Report

Collaborative Planning:

DDG1 and DDG2: collaboratively plan a route through the minefield.

Scenario ends when a workable solution has been reached to get the amphibious force to the landing area.

*Report

TAB 8

Amphibious Warfare/Minefield Avoidance using JTC and chat

Players:

BHR - MWC/DDG1

NPS - DDG2

NUWC - SSN

Initial situation:

The ESG, consisting of 3 DDGs, 1 LHD, 1 LPD, 1 LSD and 1 SSN, is now to conduct amphibious warfare, landing near port Y in order to support the Unified Force that is already assembled in the area.

The landing area is defined by: 47.00N 123.71W

47.00N 123.89W

46.90N 123.87W

46.90N 123.70W

However, one of the DDGs spotted what could have been mine-laying activity near the intended landing point. There are no other feasible places to land the force in the area though, so the possible minefield needs to be found and plotted in order to make a safe landing.

The suspected mined area roughly covers the area defined by:

46.65N 124.50W

46.65N 124.10W

47.10N 124.20W

47.10N 124.50W

An SSN is deployed with a UUV on board that could investigate the suspected mined region. The goal is to chart any found mines and plan a q-route for surface ships to go through in order to support amphibious operations.

The UUV's mission includes two rendezvous points, one relatively close to the mission area (point A at 46.70N 124.60W) and one that is well clear of the area (point B at 46.2N 124.9W). The idea being that in optimal conditions, the submarine will be able to relay any discovered mines back to the fleet faster by using point A.

Submarine will remain at periscope depth outside of visual range of sortie area and rendezvous points until time to recover UUV.

JTC:

Objects/things needed:

Initial requirements

1 SSN

2 DDG (MWC and another) for collaborative planning of Q-route

2 Rendezvous points

Track for merchant to be loitering in the area of the first rendezvous point

Items expected to be generated:

Mine laying region

Operating area for UUV

New operating area for SSN

Ability to lay out a q-route (operating area)

Deliberate Planning:

All: Create water space plan for all ships in the area to loiter before the amphibious strike. Begin each OPTASK name with the originating node (Ex: “NUWC MIO zone 1”)

BHR 0000-0800, NPS 0800-1600, NUWC 1600-2400.

NOTE: assigned times are to reduce number of current OPTASKs and clutter on the display and plans generated are completely independent.

Observer at each node type *Report in the observer channel when your node completes planning. Save OPPLAN to JC3IEDM when all nodes have completed planning.

MWC tells the submarine where to put the UUV mission using chart/map.

SSN acknowledges data.

**Report*

SSN then transmits the data to UUV.

UUV (observer at NUWC) acknowledges data input.

UUV is deployed

SSN reports UUV deployed.

Monitoring/Adaptive Planning:

5 min After the UUV is deployed, a surface ship in the area notices that a large merchant is loitering in the vicinity of rendezvous point A. (Monitor hands this information to DDG2)

DDG2 relays the information to the submarine via JTC.

<Inject solution on merchant onto chart/map of having zero speed, close to point A>

SSN orders UUV to rendezvous at point B via ACOMMS.

UUV (observer at NUWC) acknowledges change

**Report*

Rendezvous 3 min later.

After the rendezvous, the data is uploaded to the SSN.

SSN relays the mine data to the fleet via JTC both with chat and a data exchange. This information shows up on the Chart/Map as mined areas. SSN will create multiple OPTASKs illustrate the minefield.

Mined Area1:	46.73N 124.10W	Mined Area2:	46.89N 124.18W
	46.77N 124.80W		47.00N 124.26W

	46.89N 124.13W		47.02N 124.18W
	46.90N 124.06W		46.91N 124.13W
Mined Area3:	46.84N 124.23W	Mined Area4:	46.77N 124.22W
	46.89N 124.25W		46.79N 124.27W
	46.87N 124.17W		46.83N 124.19W
Mined Area5:	46.92N 124.23W		
	46.93N 124.31W		
	46.98N 124.28W		

**Report*

Collaborative Planning:

DDG1 and DDG2: collaboratively plan a route through the minefield.

Scenario ends when a workable solution has been reached to get the amphibious force to the landing area.

**Report*

APPENDIX D. DIS-XML SOURCE CODE

A. XMPPRECEIVER.JAVA

The xmppReceiver.java code is listed below.

```
 1 import java.io.*;
 2 import java.net.*;
 3 import java.util.*;
 4
 5 import org.jivesoftware.smack.*;
 6 import org.jivesoftware.smack.packet.*;
 7 import org.jivesoftware.smack.filter.*;
 8 import org.jivesoftware.smackx.muc.*;
 9
10 import org.web3d.xmsf.disutil.*;
11 import org.web3d.xmsf.dis.*;
12
13
14 /*
15  * XmppReceiver.java
16  *
17  * Created on December 7, 2005, 3:57 PM
18  *
19  *
20  */
21
22 /**
23  *
24  * @author Adrian Armold
25  */
26
27
28 public class XmppReceiver extends Object implements Runnable {
29
30     /** Default server to authenticate to */
31     public static final String DEFAULT_SERVER = "surfaris.cs.nps.navy.mil";
32
33     /** Default chat server */
34     public static final String DEFAULT_MUC_SERVER =
"conference.surfaris.cs.nps.navy.mil";
35
36     /** Default room for testing */
37     public static final String DEFAULT_ROOM = "disxml";
38
39     public static final String IADDRESS = "239.1.2.3";
40     private InetAddress multicastAddress = null;
```

```
41
42      public static final int PORT = 62040;
43
44      public MulticastSocket socket = null;
45
46      /** User name to authenticate with */
47      String username;
48
49      /** Password we use to authenticate to the server */
50      String password;
51
52      /** The server we authenticate to */
53      String authServer;
54
55      /** The muc server we join */
56      String mucServer;
57
58      /** The muc room on the muc server we joing */
59      String mucRoom;
60
61      /** Connection to the XMPP server */
62      XMPPConnection connection;
63
64      /** Multiuser chat room */
65      MultiUserChat disxml;
66
67
68
69      /** Creates a new instance of XmppReceiver */
70      public XmppReceiver (String pUsername,
71                           String pPassword,
72                           String pAuthServer,
73                           String pMucServer,
74                           String pMucRoom)
75      {
76          username = pUsername;
77          password = pPassword;
78          authServer = pAuthServer;
79          mucServer = pMucServer;
80          mucRoom = pMucRoom;
81      }
82
83          /**
84       * Establishes a connection to the XMPP server and joins the specified chat room.
85       */
86      public void login()
87      {
88  _
```

```
89 _      String mucJid = mucRoom + "@" + mucServer;
90 _      try
91         {
92             // Authenticate to our local XMPP server
93             connection = new XMPPConnection(authServer);
94             connection.login(username, password);
95 
96             // Establish a connection to the MUC room
97 
98             disxml  = new MultiUserChat(connection, mucJid);
99             disxml.join(username);
100 
101     multicastAddress = InetAddress.getByName(IADDRESS);
102     socket = new MulticastSocket(PORT);
103     socket.joinGroup(multicastAddress);
104     }
105     catch(Exception e)
106     {
107       System.out.println(e);
108     }
109 
110     // set up a packet filter to listen for only the things we want
111     PacketFilter filter = new AndFilter(new PacketTypeFilter(Message.class),
112                                         new FromContainsFilter(mucJid));
113 
114     // Next, create a packet listener. We use an anonymous inner class for brevity.
115     PacketListener myListener = new PacketListener() {
116 
117     public void processPacket(Packet ppacket)
118     {
119     if (ppacket.getProperty("disXML") == null)
120     {
121      return;
122     }
123 
124      String xmlPdu;
125      xmlPdu = (String) ppacket.getProperty("disXML");
126 
127      DisUnmarshaller unmarshaller = new DisUnmarshaller();
128      byte[] xmlToJavaBuffer = xmlPdu.getBytes();
129 
130      ByteArrayInputStream bais = new ByteArrayInputStream(xmlToJavaBuffer);
131      List pduList = new ArrayList();
132      try
133      {
134      pduList = unmarshaller.unmarshallFromXML(bais);
135      }
136      catch(Exception e)
```

```
137       {
138       System.out.println(e);
139       }
140
141        /* This needs stream connection information to drop the binary DIS on the wire.
142         * Taken from WriterExample.java authored by Don McGregor */
143
144       Iterator pduIterator = pduList.iterator();
145       while (pduIterator.hasNext())
146       {
147
148       EntityStatePdu pdu = (EntityStatePdu)pduIterator.next();
149
150       try
151       {
152       DisMarshaller disMarshaller; // Puts Java objects into binary DIS format
153       DatagramPacket packet;       // The UDP/Multicast packet we will send
154       byte javaToDisBuffer[];      // Holds binary format DIS packet
155
156       disMarshaller = new DisMarshaller();
157
158       // disMarshaller puts the espdu into the correct binary format
159       javaToDisBuffer = disMarshaller.marshallPduToSend((EntityStatePdu)pdu);
160
161       // And create a UDP packet to the default destination
162       packet = new DatagramPacket(javaToDisBuffer, javaToDisBuffer.length,
multicastAddress, PORT);
163
164       // And finally send it.
165       socket.send(packet);
166       }
167       catch(Exception e)  // Catch-all exception
168       {
169       System.out.println(e);
170       }
171       } // End while loop
172
173       try
174       {
175       Message message = disxml.createMessage();
176       message.setBody("A DISXML message has been received and processed.");
177       disxml.sendMessage(message);
178       } catch(Exception e)
179       {
180       System.out.println(e);
181       }
182
183
```

```
184      } // end process packet
185
186      };// end PacketListener anonymous class
187      connection.addPacketListener(myListener,filter);
188
189      }
190
191
192      public void run()
193     {
194         while (true) {
195             try {
196                 Thread.sleep(100);
197             } catch (InterruptedException ie) {
198     }
199
200
201         }
202     }
203
204
205     /**
206      * @param args the command line arguments
207      */
208     public static void main(String[] args) {
209          XmppReceiver receiver =  new XmppReceiver("snerd1", "xmpp",
210                                              DEFAULT_SERVER,
211                                              DEFAULT_MUC_SERVER,
212                                              DEFAULT_ROOM);
213         receiver.login();
214         receiver.run();
215     }
216
217 }
```

B. XMPPSENDER.JAVA

The xmppSender.java code is listed below.

```
 2 import java.io.*;
 3 import java.util.*;
 4
 5 import org.jivesoftware.smack.*;
 6 import org.jivesoftware.smack.packet.*;
 7 import org.jivesoftware.smack.filter.*;
 8 import org.jivesoftware.smackx.muc.*;
 9
10 import org.web3d.xmsf.disutil.*;
```

```
11 import org.web3d.xmsf.dis.*;
12
13 /**
14  * A Runnable (threadable) objet that sends traffic to an XMPP multi-user chat
15  * room, and receives traffic as well. Note that there are two techniques for
16  * sending messages, one with a property attachment and one in a "standard" way
17  * that will show up in most XMPP client chat rooms.
18  *
19  * @author DMcG
20  */
21 public class XMPPSender extends Object implements Runnable
22 {
23        /** Default server to authenticate to */
24     public static final String DEFAULT_SERVER = "surfaris.cs.nps.navy.mil";
25
26     /** Default chat server */
27     public static final String DEFAULT_MUC_SERVER =
"conference.surfaris.cs.nps.navy.mil";
28
29     /** Default room for testing */
30     public static final String DEFAULT_ROOM = "disxml";
31
32     /** User name to authenticate with */
33     String sender;
34
35     /** Password we use to authenticate to the server */
36     String password;
37
38     /** The server we authenticate to */
39     String authServer;
40
41     /** The muc server we join */
42     String mucServer;
43
44     /** The muc room on the muc server we joing */
45     String mucRoom;
46
47     /** Connection to the XMPP server */
48     XMPPConnection connection;
49
50     /** Multiuser chat room */
51     MultiUserChat disxml;
52
53     /** Frequency of messages */
54     long frequency;
55
56     /**
57      * Constructor
```

```
58      */
59     public XMPPSender(String pUsername,
60                       String pPassword,
61                       String pAuthServer,
62                       String pMucServer,
63                       String pMucRoom,
64                       long   pFrequency)
65     {
66         sender = pUsername;
67         password = pPassword;
68         authServer = pAuthServer;
69         mucServer = pMucServer;
70         mucRoom = pMucRoom;
71         frequency = pFrequency;
72     }
73
74     /**
75      * Establishes a connection to the XMPP server and joins the specified chat room.
76      */
77     public void login()
78     {
79         try
80         {
81             // Authenticate to our local XMPP server
82             connection = new XMPPConnection(authServer);
83             connection.login(sender, password);
84
85             // Establish a connection to the MUC room
86             String mucJid = mucRoom + "@" + mucServer;
87             disxml  = new MultiUserChat(connection, mucJid);
88             disxml.join(sender);
89
90         }
91         catch(Exception e)
92         {
93             System.out.println(e);
94         }
95     }
96
97     public void run()
98     {
99
100        try
101        {
102
103            System.out.println("Beginning to send messages");
104
105            PduFactory pduFactory = new PduFactory();
```

```
106             EntityStatePduType espdu;
107
108             // Create an ESPDU to send
109             espdu = pduFactory.getEntityStatePdu();
110
111             // Set the entity ID, the unique identifer of an entity in the world.
112             EntityIDType id = espdu.getEntityID();   // Globally unique identifier
for the entity
113             id.setSite(0);
114             id.setApplication(1);
115             id.setEntity(2);
116
117             // "default" length of an entity state PDU with no articulation
parameters--144 bytes
118             espdu.getPduHeader().setLength(144);
119
120             while(true)
121             {
122                 for(int idx = 0; idx < 100; idx++)
123                 {
124                     // Modify the position of DIS packet here...
125                      Vector3Double location = espdu.getEntityLocation();
126
127                       // update location
128                       location.setX((double)idx);
129
130
131                     // Marshall out the packet to XML
132                     DisMarshaller marshaller = new DisMarshaller();
133                     ByteArrayOutputStream baos = new ByteArrayOutputStream();
134
135                     ArrayList pduList = new ArrayList();
136                     pduList.add(espdu);
137                     marshaller.marshallPdus(pduList, baos);
138                     String xmlPduString = baos.toString();
139
140                     // Create an XMPP message and attach the DIS-XML xml to it
141                     Message message = disxml.createMessage();
142                     message.setProperty("disXML", xmlPduString);
143                     message.setBody("A DISXML message from the sender.");
144
145
146                     // The properties method--send the message
147                     disxml.sendMessage(message);
148
149                         // Sleep for a random amount of time, then send another.
150                   double ran = Math.random();
151                   long sleepTime = (long)(ran * frequency);
```

```
152                   Thread.sleep(sleepTime);
153
154                }
155            }
156
157            // connection.close();
158            }
159
160        catch(Exception e)
161        {
162            System.out.println(e);
163        }
164    }
165
166    /**
167     * entry point
168     */
169    public static void main(String args[])
170    {
171
172        XMPPSender sender =  new XMPPSender("snerd0", "xmpp",
173                                           DEFAULT_SERVER,
174                                           DEFAULT_MUC_SERVER,
175                                           DEFAULT_ROOM,
176                                           1000);
177        sender.login();
178        sender.run();
179
180    }
181 }
```

```
173    try
174    {
175     Message message = disxml.createMessage();
176     message.setBody("A DISXML message has been received and processed.");
177     disxml.sendMessage(message);
178    } catch(Exception e)
179    {
180     System.out.println(e);
181    }
```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX E. XTC CODEBASE

The XTC code base is located at the NPS sourceforge site under XMSF.

It can be downloaded at: http://xmsf.cvs.sourceforge.net/xmsf

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF REFERENCES

Barrett, D. 2006, *Carrier Strike Group Twelve Sponsors Fleet/Joint/Coalition Testing of Open Standards Chat Tool.* [Electronic version]. CHIPS - the Department of the Navy Information Technology Magazine, (January-March 2006). Retrieved 01 September 2006.

Bos, B. (2000). *XML in 10 points.* Retrieved 18 July 2006 from http://www.w3.org/XML/1999/XML-in-10-points.

Bouret, R. (1999). *XML and Databases.* Retrieved 16 August 2006 from http://www.rpbourret.com/xml/XMLAndDatabases.htm#query.

Brutzman, D., McGregor, D., DeVos, D. A., Lee, C. S., Amsden, S., and Blais, C., et al. (2004). *XML-Based Tactical Chat (XTC): Requirements, Capabilities and Preliminary Progress* (Technical Report No. NPS-MV-2004-001). Monterey, California: Naval Postgraduate School.

Brutzman, D. (2003). *Web-Based 3D Graphics Rendering of Dynamic Deformation Structures in Large-Scale Distributed Simulations.* (Technical Report No. NPS-MV-2003-01). Monterey, California: Naval Postgraduate School.

Burns, T. (2006). *Coalition Secure Management and Operations System (COSMOS) ACTD.* Defense Information Systems Agency. Retrieved 20 September 2006 from http://www.les.disa.mil/c/extranet/home?e_l_id=32..

Chaum, E. (2006a). *JC3IEDM-Enhanced Tactical Collaboration (JTC).* Unpublished manuscript. Retrieved 07 September 2006.

Chaum, E. (2006b). *JC3IEDM-Enhanced Tactical Collaboration (JTC) Quick-Look Report.* Unpublished manuscript. Retrieved 07 September 2006.

Cokus, M., and Pericas-Geertsen, S. (2005). *XML Binary Characterization Use Cases W3C Working Group Note.* 31 March 2005 W3C. Retrieved 16 August 2006.

Eovito, B. A. (2006). *An Assessment of Joint Chat Requirements from Current Usage Patterns.* (Master's Thesis, Naval Postgraduate School).

Fletcher, B., Lirrette, K., and Bishop, M. (2006). *Cross Domain Collaborative Information Environment (CDCIE) Collaboration Tool Overview.* Unpublished manuscript. Retrieved 26 August 2006.

Goldman, O., Berjon, R., and Bournez, C. (2005). *Charter of the Efficient XML Interchange Working Group.* W3C. Retrieved 16 August 2006.

Gospodnetic, O., Hatcher, E. (2005). *Lucene in Action.* Manning Publications Company, Greenwich, Connecticut.

Hildebrand, J. (2003). *Nine IM Accounts and Counting*. [Electronic version]. *Queue, 1*(8), 44-50.

Hildebrand, J., and Saint-Andre, P. (2004). *JEP-0080: User Geolocation.* Jabber Software Foundation. Retrieved 23 August 2006.

IETF, Network Working Group. (2004). *Extensible Messaging and Presence Protocol (XMPP):Core.* Retrieved 18 July 2006 from http://www.ietf.org/rfc/rfc3920.txt?number=3920.

Jive Software. (2006). *Smack Documentation.* Retrieved 29 August 2006, 2006 from http://www.jivesoftware.org/builds/smack/docs/latest/documentation/index.html.

Kalt, C. (2000a). *Internet Relay Chat: Channel Management (RFC 2811).* [Electronic version]. *IETF, RFC 2811.* Retrieved 18 September 2006.

Kalt, C. (2000b). *Internet Relay Chat: Client Protocol (RFC 2812).* [Electronic version]. *IETF RF, 2812.* Retrieved 18 September 2006.

Kalt, C. (2000c). *Internet Relay Chat: Server Protocol (RFC 2813).* [Electronic version]. *IETF RFC 2813.* Retrieved 18 September 2006.

Kalt, C. (2000d). *RFC 2810 Internet Relay Chat.* [Electronic version]. *Architecture,* Retrieved 18 September 2006.

Kell, J. (1987). *Relay: Past, Present, and Future*. Paper presented at the *Spring NETCON 1987*, New Orleans. Retrieved 20 July 2006 from http://web.inter.nl.net/users/fred/relay/relhis.html.

McGregor, D., Brutzman, D., Armold, A., and Blais, C. (2006). *DIS-XML: Moving DIS to Open Data Exchange Standards*. SISO, 2006 Spring Simulation Interoperability Workshop, Huntsville, Alabama.

Millard, P., Saint-Andre, P., and Meijer, R. (2005). *JEP-0060: Publish-Subscribe.* Jabber Software Foundation. Retrieved 23 August 2006.

Molitoris, J.J. (2003). *Use of COTS XML and Web Technology for Current and Future C2 Systems.* Paper presented at Military Communications Conference, 2003, Boston, Massachusetts.

Multilateral Interoperability Programme. (2006). *Multilateral Interoperability Programme: Background and History*. Retrieved 28 July 2006 from http://www.mip-site.org/020_Public_History.htm.

Oikarinen, J. (2005). *IRC History*. Retrieved 20 July 2006 from http://www.irc.org/history_docs/jarkko.html.

Oikarinen, J., and Reed, D. (1993). *Internet Relay Chat (IRC) Protocol*. [Electronic version]. *IETF RFC 1459*. Retrieved 18 September 2006.

Paterson, I., Saint-Andre, P., and Smith, D. (2006). *JEP-0116: Encrypted Sessions.* Jabber Software Foundation. Retrieved 23 August 2006.

Pericas-Geertsen, S. (2003). *Binary Interchange of XML Infosets*. Paper presented at the *XML Conference and Exposition 2003 - Proceedings,* Philadelphia, PA. Retrieved 16 August 2006.

Saint-Andre, P. (2005). *JEP-0045: Multi-User Chat.* Jabber Software Foundation. Retrieved 21 August 2006.

Saint-Andre, P. (2004a). *Extensible Messaging and Presence Protocol (XMPP): Core.* Retrieved 21 August 2006.

Saint-Andre, P. (2004b). *RFC 3921: Extensible Messaging and Presence Protocol (XMPP): Instant Messaging and Presence.* IETF RFC 3921. Retrieved 21 August 2006.

Saint-Andre, P., and Meijer, R. (2005). *Streaming XML with Jabber/XMPP*. Internet Computing, IEEE, 9(5), 82-89.

Sanders, T. (2006). *Yahoo and MSN Marry IM Services.* Retrieved 20 July 2006 from http://www.vnunet.com/vnunet/news/2143773/yahoo-msn-marry-messengers.

Smith, D., Saint-Andre, P., and Paterson, I. (2005). *JEP-0124: HTTP Binding.* Jabber Software Foundation. Retrieved 1 September 2006.

USJFCOM. (2003). *Deployable Joint Command and Control System (DJC2 Baseline Requirements Document (BRD) Version 1.0* . Suffolk, VA: USJFCOM.

W3C Press Release (1997), *W3C Issues XML1.0 as a Proposed Recommendation*. W3C. Retrieved 20 July 2006.

Wikipedia.org. (2006a). *AOL Instant Messenger.* Retrieved 20 July 2006 from http://en.wikipedia.org/wiki/AOL_Instant_Messenger.

Wikipedia.org. (2006b). *Instant Messaging.* Retrieved 11 September 2006 from http://en.wikipedia.org/wiki/Instant_messaging.

Wikipedia.org. (2006c). *Talk (Unix).* Retrieved 11 September 2006 from http://en.wikipedia.org/wiki/Unix_talk.

Woods, R. (2005). *Trident Warrior Experiment Series*. San Diego, CA: SPAWAR. Retrieved 18 September 2006, from www.enterprise.spawar.navy.mil/getfile.cfm?contentId=400&type=C.

Web3d.org. (2006). *X3D and Related Specifications.* Retrieved 20 September 2006 from http://www.web3d.org/x3d/specifications/#x3d-spec.

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California

3. Marine Corps Representative
   Naval Postgraduate School
   Monterey, California

4. Dr. Don Brutzman
   Naval Postgraduate School
   Monterey, California

5. Don McGregor
   Naval Postgraduate School
   Monterey, California

6. Terry Norbraten
   Naval Postgraduate School
   Monterey, California

7. Dr. Peter Denning
   Naval Postgraduate School
   Monterey, California

8. Boyd Fletcher
   United States Joint Forces Command J9
   Suffolk, Virginia

9. Monica Shephard
   United States Joint Forces Command J9
   Suffolk, Virginia

10. RADM James Winnefeld
    United States Joint Forces Command J9
    Suffolk, Virginia

11. Erik Chaum
    Naval Undersea Warfare Center
    Newport, Rhode Island

12. Fred Burkley
Naval Undersea Warfare Center
Newport, Rhode Island

13. Dr. LorRaine Duffy
Space and Naval Warfare Systems Center
San Diego, California

14. Dr. Shelley Gallup
Naval Postgraduate School
Monterey, California

15. Captain Adrian D. Armold
United States Marine Corps
Seaside, California

16. Dr. Mel Armold
Alamosa, Colorado

17. Director, Training and Education, MCCDC, Code C46
Quantico, Virginia

18. Director, Marine Corps Research Center, MCCDC, Code C40RC
Quantico, Virginia

19. Marine Corps Tactical Systems Support Activity (Attn: Operations Officer)
Camp Pendleton, California

20. Deputy Commander, C4I Integration (SG 06)
Marine Corps Systems Command
Quantico, Virginia

21. Director, Systems Engineering and Integration (SG 061)
Marine Corps Systems Command
Quantico, Virginia

22. Director, Information Assurance and Joint Requirements (SG 062)
Marine Corps Systems Command
Quantico, Virginia

23. Operations Manager, Systems Engineering and Integration
Marine Corps Systems Command
Quantico, Virginia

24. Technical Director
Marine Corps Technical Systems Support Activity
Camp Pendleton, California

25. Dianne Boettcher
SRA International
Fairfax, Virginia

26. USW-XML Mailing List
Monterey, California

27. Captain Richard Lee USN Retired
Office of the Secretary of Defense
Washington, District of Columbia

28. RADML Bill Landay III USN
Office of Naval Research
Arlington, Virginia

29. Captain James Neushul
First Light Armored Reconnaissance Battalion
Camp Pendleton, California

30. Dr. Dan Boger
Naval Postgraduate School
Monterey, California

31. Tom Burns
Defense Information Systems Agency
Arlington, Virginia

32. Dr. Yvonne Masakowski
Naval Undersea Warfare Center
Newport, Rhode Island

33. Pierre Corriveau
Naval Undersea Warfare Center
Newport, Rhode Island

34. Captain David G. Yoshihara
U.S Pacific Fleet
Pearl Harbor, Hawaii

35. Captain Scott Miller USN
Space and Naval Warfare Systems Center
San Diego, California

36. USW- Announce Mailing List
Naval Postgraduate School
Monterey, California

37. XMSF-Announce Mailing List
Naval Postgraduate School
Monterey, California

38. XTC Mailing List
Naval Postgraduate School
Monterey, California

39. F.P. Gustavson
Kailua, Hawaii

40. Commander Danelle Barrett USN
Pacific Command
Camp Smith, Hawaii

41. Chris Gunderson
W2COG
Reston, Virginia

42. Eugene Hackney
U.S. Pacific Fleet
Pearl Harbor, Hawaii

43. Richard Coupland
Naval Sea Systems Command
Newport, Rhode Island

44. Mark Kenny
Submarine Group Two
Groton, Connecticut