2004-01-28

XML-based tactical chat (XTC): requirements, capabilities and preliminary progress

Brutzman, Don

Naval Postgraduate School (U.S.)

http://hdl.handle.net/10945/47558

NAVAL
POSTGRADUATE
SCHOOL

MONTEREY CALIFORNIA

TECHNICAL REPORT

Approved for public release; distribution is unlimited.

XML-BASED TACTICAL CHAT (XTC): REQUIREMENTS, CAPABILITIES AND PRELIMINARY PROGRESS

by

Don Brutzman, Don McGregor,
Daniel A. DeVos and Chin Siong Lee
with
Saundra Amsden, Curtis Blais, Duane Davis,
Dimitrios Filiagos and Brian Hittner

28 January 2004
Technical Report NPS-MV-2004-001

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** January 2003	**3. REPORT TYPE AND DATES COVERED** Technical Paper

4. TITLE AND SUBTITLE: **XML-BASED TACTICAL CHAT (XTC): REQUIREMENTS, CAPABILITIES AND PRELIMINARY PROGRESS**	**5. FUNDING NUMBERS**
6. AUTHORS Don Brutzman, Don McGregor, Daniel DeVos and Chin Siong Lee, with Saundra Amsden, Curtis Blais, Duane Davis, Dimitrios Filiagos and Brian Hittner	
7. PERFORMING ORGANIZATION NAME AND ADDRESS MOVES Institute, Naval Postgraduate School Monterey, CA 93943-5000	**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING /MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A	**10. SPONSORING/MONITORING AGENCY REPORT NUMBER**

11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the authors and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited.	**12b. DISTRIBUTION CODE**

13. ABSTRACT

The motivation for pursuing XML-based tactical chat includes the great potential of this technology and fixing limitations of current chat programs. XTC capabilities have the potential to completely upgrade and restructure all tactical military communications. The current tools for military chat include IRC, Yahoo, MSN, AIM, ICQ, and NKO. None of these provides the full functionality or interoperability needed in a joint environment. Moreover, if a nonproprietary chat protocol is developed, it can lead to a decision-support environment in which data, text, audio, and video can be logged, evaluated and managed, all in a Web environment where no additional specialized software or hardware is needed.

Chat technology challenges for the military fit into three areas: tactical, technical, and administrative. Tactically, there are many ways chat can be used, but effective practices are not yet defined in procedures or doctrine. Joint forces use a myriad of chat programs that don't interoperate and are usually proprietary. Technically, many chat programs are barred by firewalls and lack a robust interface to allow logging and searching past chats. From an administrative prospective, plain-text chat has no structure. Scheduling and controlling who attends or converses remains undefined. Within DoD there is no standard for how, when, and by whom chats ought to be conducted.

Possible approaches to these problems include adopting a proprietary chat system or customizing an open-source implementation. Proprietary solutions are costly, do not interoperate well, and are too inflexible for a technology that is evolving rapidly. Open-source software can provide a solution that is adaptable, extensible, quick to implement, straightforward to maintain, and relatively inexpensive.

This report provides a preliminary assessment of XML-based tactical chat (XTC) using an open-source, open-standards solution. Promising initial results demonstrate that an XML document can be sent from a XHTML page in a Web browser to an off-the-shelf Jabber client via a Web server. Further, available server and client implementations can enable a research and development plan for rapid development. Further work on XTC as part of the Extensible Modeling and Simulation Framework (XMSF) is justified and needed.

14. SUBJECT TERMS tactical chat, Jabber, XTC, XMSF, XML, XHTML	**15. NUMBER OF PAGES**
	16. PRICE CODE

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

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

The motivation for pursuing XML-based tactical chat includes the great potential of this technology and fixing limitations of current chat programs. XTC capabilities have the potential to completely upgrade and restructure all tactical military communications. The current tools for military chat include IRC, Yahoo, MSN, AIM, ICQ, and NKO. None of these provides the full functionality or interoperability needed in a joint environment. Moreover, if a nonproprietary chat protocol is developed, it can lead to a decision-support environment in which data, text, audio, and video can be logged, evaluated and managed, all in a Web environment where no additional specialized software or hardware is needed.

Chat technology challenges for the military fit into three areas: tactical, technical, and administrative. Tactically, there are many ways chat can be used, but effective practices are not yet defined in procedures or doctrine. Joint forces use a myriad of chat programs that don't interoperate and are usually proprietary. Technically, many chat programs are barred by firewalls and lack a robust interface to allow logging and searching past chats. From an administrative prospective, plain-text chat has no structure. Scheduling and controlling who attends or converses remains undefined. Within DoD there is no standard for how, when, and by whom chats ought to be conducted.

Possible approaches to these problems include adopting a proprietary chat system or customizing an open-source implementation. Proprietary solutions are costly, do not interoperate well, and are too inflexible for a technology that is evolving rapidly. Open-source software can provide a solution that is adaptable, extensible, quick to implement, straightforward to maintain, and relatively inexpensive.

This report provides a preliminary assessment of XML-based tactical chat (XTC) using an open-source, open-standards solution. Promising initial results demonstrate that an XML document can be sent from a XHTML page in a Web browser to an off-the-shelf Jabber client via a Web server. Further, available server and client implementations can enable a research and development plan for rapid development. Further work on XTC as part of the Extensible Modeling and Simulation Framework (XMSF) is justified and needed.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

LIST OF FIGURES

ACKNOWLEDGMENTS

This report provides an initial assessment of requirements, capabilities, and proof-of-concept application for Extensible Markup Language (XML) -based tactical chat (XTC). Preliminary work was produced as a quarter-long class project during graduate course MV4205, “Advanced XML.” Margaret Davis provided technical editing support. All student and staff contributions are gratefully acknowledged.

Erik Chaum of the Naval Undersea Warfare Center (NUWC) provided inspiration for pursuing XTC through his experiences with fleet-based chat during virtual submarine operations on a shore-based virtual nuclear-powered guided missile submarine (VSSGN) during fleet battle experiments in 2001.

Extensible Markup and Simulation Framework (XMSF) partner Mark Pullen of George Mason University (GMU) provided excellent insights into application and adaptation of chat-distribution algorithms with respect to selectively reliable multicast routing.

Rich Goldberg of Aniviza, Inc. and Alan Hudson of Yumetech, Inc. assisted in establishing a Jabber-based chat network for XTC experimentation and support of the X3D developer community.

Bruce DeLagi and Jim McGuinness of Sun Microsystems collaborated on the possible integration of Java-card access control and security services in a combination of secure and non-secure environments.

Anthony Ceri and the staff of Joint Forces Command provided encouragement and support for pursuing XTC during distributed, continuous-experiment environment exercises (DCEE) featuring XMSF technologies during the summer of 2003.

Jabber.org and the Jabber community provide an open-source collaboration framework for a wide range of public and private Jabber servers, open-source projects, and software companies, all using the Jabber protocol to build real-time applications and services. These resources are critical to future progress.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

A. PROBLEM STATEMENT

Text-based chat communications are used daily by the military to pass information, coordinate operations, and support collaborative decision-making. There is currently no standard, however, for application features, rules of conduct, or guidance on monitoring and capturing the information that transpires in chats. A standardized chat system has the potential to handle large volumes of information that can be organized and disseminated for decision support.

Extensible Markup Language (XML) provides a framework for standardizing data formats that is applicable to a wide variety of information types, including chat communications. Jabber is an open-source, extensible chat protocol that provides an excellent chat framework. This report shows that Jabber-driven, XML-based tactical chat (XTC) can provide an excellent environment for military communication, supporting both current needs and future decision-support capabilities. At the same time, Jabber-based XTC avoids proprietary client restrictions that limit effectiveness, reliability, security, and interoperability.

B. REQUIREMENTS OVERVIEW

The requirements for an effective tactical military chat system fall into in three major areas: the tactical, technical, and administrative.

An examination of tactical requirements begins with defining when and why chats are conducted in a military setting. As the benefits and shortcomings of current programs are reviewed, future chat systems can capitalize on strengths, avoid weaknesses, and identify potential features that will take chat to a new level of benefit. Chat is routinely used for command and control, operational planning and execution, and analysis; nevertheless, none of these is defined in current communication doctrine, nor do these uses exploit the full potential of chat.

The technical requirements for military chat arise from the unique nature of the communications environment. Technical capabilities of interest include asynchronous protocol, communicating across strict firewall settings, joint compatibility, multiple message types, verifying code reliability, and designing adaptable interfaces. Many of these data-interchange requirements are met by XML. Of particular importance is that the issues involved in exchanging, organizing, storing, and analyzing large volumes of chat information can be solved under XML.

Administrative requirements need also to be considered, because chat is a relatively new coordination tool for military use. Lessons learned from large group discussions can be used to define user roles and rules of conduct, so that time spent in chat is well used. Scheduling, rendezvousing, monitoring, and controlling access are administrative requirements that will improve chat effectiveness. The relationships between decentralized chat rooms exploiting tactical opportunity and formal, chain-of-command tactical communications needs to be better understood. Through effective guidelines, chat can become a immensely valuable tool for personnel.

This report provides a preliminary assessment of chat issues and opportunities, focused on tactical, technical, and administrative requirements. An explanation of Jabber technology and how it can be deployed is included, as well as promising initial-implementation results. One demonstration shows that an XML document can be sent from a XHTML page in a Web browser to a chat room, using open-standards, open-source Jabber clients and Web servers. In addition it provides a base server and client implementation and a strategic R&D plan for rapid development of XML-based military chat (referred to as “tactical chat”).

C. MOTIVATION

The motivation for pursuing XML-based tactical chat (XTC) was the desire to offer the power of chat across the spectrum of tactical operations. Such potential is inspired by the large-scale growth of commercial chat clients, the XML capabilities of Jabber-based chat, and the obvious potential of XML technology.

The U.S. Joint Forces Command (JFCOM) is working with the Space and Naval Warfare Systems Command (SPAWAR) and the Defense Collaboration Tool Suite 3.0 engineering team to replace Envoke (a secure messaging and presence platform developed as a research project within DoD) with the Jabber/XMPP (extensible messaging and presence protocol). Dissatisfaction with Envoke includes poor performance and reliability, and its proprietary protocol. Additionally, extending the Jabber/XMPP protocol to meet several military needs is being explored, including:

1) Cross-domain chat and collaboration

2) Secure, low-cost, high-performance, reliable, textual chat

3) Chat with translation (i.e., English to/from other languages)

4) Scalable standards-based replacement for T.120, Envoke, and much of IWS that will reduce the procurement and deployment costs for collaboration tools for the DoD by an order of magnitude. [Boyd 03]

Other tools presently used for chat in the military include Yahoo, MSN, AIM, ICQ, and NKO. None provides the full functionality, security, or interoperability needed in a joint environment. XMPP can be extended via XML namespaces using the Jabber enhancement proposal process to support:

1) Whiteboarding

2) Audio

3) Video

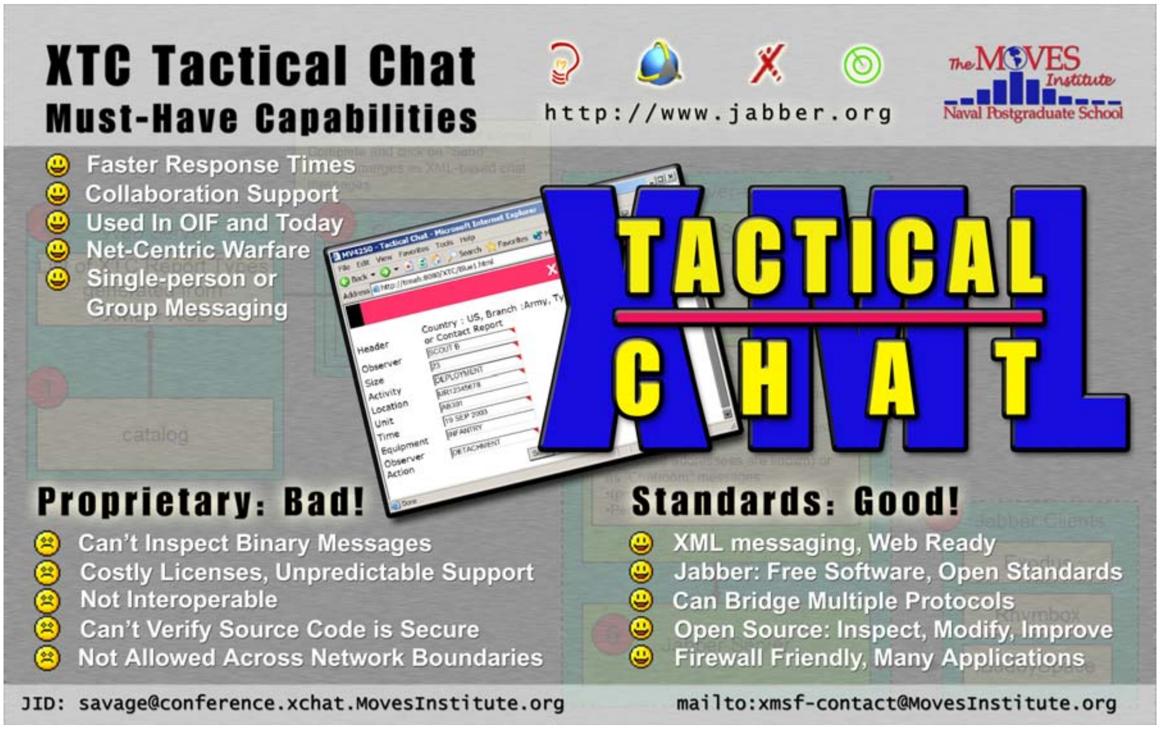
4) Translation

5) Message labeling and content signing

6) Application casting (1-way sharing)

If adopted and approved, these capabilities may enable a decision-support environment that can be logged and managed and works in a Web environment with no other specialized software or hardware than commonly installed Web technology.

Figure 1 shows the XTC poster presented at the Interservice/Industry Training, Simulation and Education Conference (I/ITSEC) in 2003, summarizing the capabilities and benefits of an XTC approach. Figure 2 restates XTC must-have capabilities, problems with proprietary applications, and standards benefits in text form.



XTC Tactical Chat
Must-Have Capabilities

http://www.jabber.org

The MOVES Institute
Naval Postgraduate School

- Faster Response Times
- Collaboration Support
- Used In OIF and Today
- Net-Centric Warfare
- Single-person or Group Messaging

Proprietary: Bad!

- Can't Inspect Binary Messages
- Costly Licenses, Unpredictable Support
- Not Interoperable
- Can't Verify Source Code is Secure
- Not Allowed Across Network Boundaries

Standards: Good!

- XML messaging, Web Ready
- Jabber: Free Software, Open Standards
- Can Bridge Multiple Protocols
- Open Source: Inspect, Modify, Improve
- Firewall Friendly, Many Applications

JID: savage@conference.xchat.MovesInstitute.org

mailto:xmsf-contact@MovesInstitute.org

Figure 1. XML-based tactical chat (XTC) poster presented at I/ITSEC 2003 summarizes required features, shortfalls of commercial technologies and the benefits of open standards.

Must-Have Capabilities

- Test Faster response times
- Collaboration support
- Used in Operation Iraqi Freedom (OIF) and today
- Net-centric warfare
- Single-person or group messaging

Problems with proprietary solutions

- Can’t inspect binary messages
- Costly licenses, unpredictable support
- Not interoperable
- Can’t verify source code is secure
- Not allowed across network boundaries

Benefits of standards-based solutions

- XML messaging, Web ready
- Jabber: Free software, open standards
- Can bridge multiple protocols
- Open source: inspect, modify, improve
- Firewall friendly, many applications

Figure 2. XTC must-have capabilities, proprietary problems, and standards benefits.

D. OBJECTIVES

To establish specific requirements for XTC use in military environments, this report addresses the following research questions:

- What benefits can XML-based tactical chat provide to military communications?
- What are the tactical requirements for XTC?
- What are the technical requirements for XTC?
- What are the administrative requirements for XTC?
- How can XML be applied to chat?
- What is Jabber and how can it be used for tactical chat?

- What open-source Jabber clients are available and how are they implemented?
- Can XML be sent from a client to a Jabber server and displayed?
- Can chat information be sent using a thin client (XHTML page) instead of thick (Jabber client)?
- Can XML-based chat also be used for software-agent and combat-control systems (CCS) communications?
- How is a Jabber server configured?
- What are the security benefits and concerns under Jabber?

E. DOCUMENT ORGANIZATION

Chapter I provides a requirements overview, motivation, and report objectives for XML-based Tactical Chat (XTC). Chapter II reviews related work in multicast networking, 3D graphics, large-scale virtual environments, and the Extensible Modeling and Simulation Framework (XMSF). Chapter III provides a list of tactical, technical, and administrative requirements needed for tactical chat. Chapter IV is an overview of Jabber technology. Chapter V is a Jabber development guide, providing details on how to set up a Jabber client and server, also integrating servlets and Java bean capabilities. Chapter VI outlines the initial design, testing, and lessons learned for the preliminary XTC application produced during this project. Chapter VII examines security concerns and considerations. Chapter VIII summarizes conclusions and gives recommendations for future work. The appendices provide exemplar XML schema, style sheets, server code, and references.

II. RELATED WORK

A. INTRODUCTION

The chapter provides an overview of diverse topics that provide information relevant to XTC. Topics covered include multicast backbone (MBone), selective reliability multicast, extensible 3D (X3D) graphics, and large-scale virtual environments (LSVEs). The Extensible Modeling and Simulation Framework (XMSF), distributed interactive simulation (DIS) protocol and the DIS-Java-VRML (Virtual Reality Modeling Language) project, and XML binary serialization using the Cross-Format Schema Protocol (XFSP) are presented as well.

B. MULTICAST BACKBONE (MBONE)

Most multicasting packets are prevented from crossing network boundaries, such as routers, because of their potential to saturate networks. The multicast backbone (MBone) is a technology that enables the distribution of live audio, video, and other packets on a global scale. MBone controls multicast-packet distribution by limiting their lifetime and adaptively restricting multicast transmissions via sophisticated pruning algorithms, while allowing multicast packets to "tunnel" through IP routers. [Macedonia, Brutzman 94]

MBone is a virtual network because it shares the same physical media as the Internet. It uses a network of routers (mrouters) that can support multicast. [Brutzman 03] Of particular interest to XTC is the ability to allow real-time video, audio and whiteboarding in a low-bandwidth environment. Unfortunately, wide-area multicast is typically difficult to implement in practice. Chat channels provide a valuable alternative.

C. SELECTIVELY RELIABLE MULTICAST

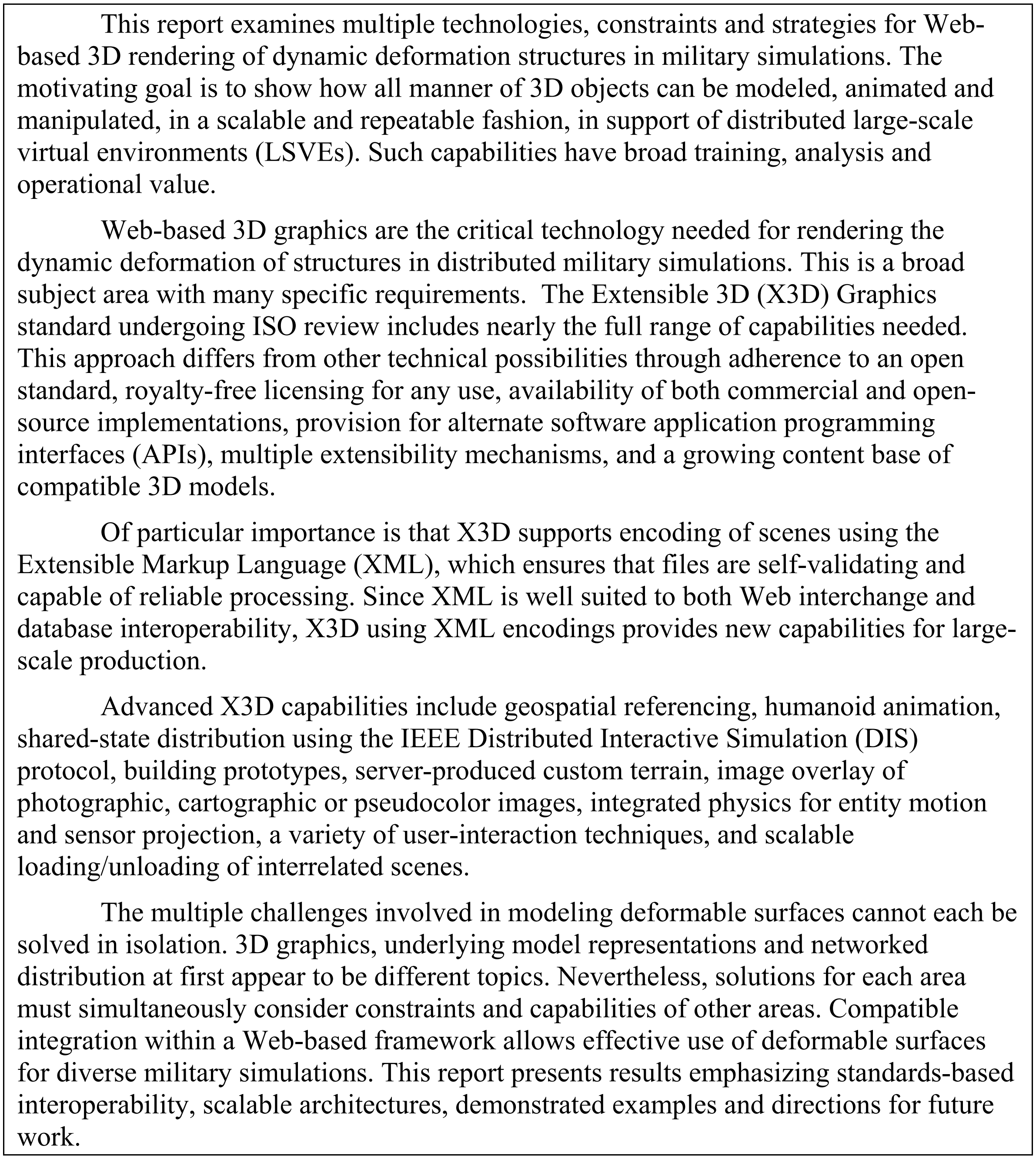
Selectively reliable multicast uses the selectively reliable transport protocol (SRTP) to expand the transmission-control protocol's (TCP) unicast address and port multiplexing structure to multicast groups and simulation-object classes (sometimes called "categories"). The initial implementation of SRTP is to support distributed virtual simulations (DVS) that are distributed over a network, operate with human participants, and require virtual representation in real time. This requires that the interconnecting network deliver messages with minimal packet loss and low latency (delay). [Pullen 99] The traditional TCP connection is always point to point and does not support multicast.

Similar to the military tactical environment, "network-based distributed simulation characteristically generates large amounts of message traffic among the computers supporting elements of the simulation. In the general case this requires many-to-many communications among a group of computers conducting some aspect of the simulation." [Pullen 99] There is potential to apply SRTP to XTC in order to harness its benefits of minimal delay in a high-traffic environment.

D. X3D GRAPHICS

The X3D specification uses XML to encode 3D scenes; specifically, the geometries and behaviors expressed in the VRML 97 encoding. Because X3D is coded as XML, XTC clients might use the X3D format to pass virtual-reality models among participants. Jabber chat capabilities might also be integrated into X3D script nodes as a streaming mechanism for animation data, thus providing a many to many channel for participants in virtual environments. These are important areas for future work.

Figure 3 contains the abstract from *Web-Based 3D Graphics Rendering of Dynamic Deformation Structures in Large-Scale Distributed Simulations* [Brutzman 2003] as a detailed reference showing the value real-time web-based 3D graphics.



This report examines multiple technologies, constraints and strategies for Web-based 3D rendering of dynamic deformation structures in military simulations. The motivating goal is to show how all manner of 3D objects can be modeled, animated and manipulated, in a scalable and repeatable fashion, in support of distributed large-scale virtual environments (LSVEs). Such capabilities have broad training, analysis and operational value.

Web-based 3D graphics are the critical technology needed for rendering the dynamic deformation of structures in distributed military simulations. This is a broad subject area with many specific requirements. The Extensible 3D (X3D) Graphics standard undergoing ISO review includes nearly the full range of capabilities needed. This approach differs from other technical possibilities through adherence to an open standard, royalty-free licensing for any use, availability of both commercial and open-source implementations, provision for alternate software application programming interfaces (APIs), multiple extensibility mechanisms, and a growing content base of compatible 3D models.

Of particular importance is that X3D supports encoding of scenes using the Extensible Markup Language (XML), which ensures that files are self-validating and capable of reliable processing. Since XML is well suited to both Web interchange and database interoperability, X3D using XML encodings provides new capabilities for large-scale production.

Advanced X3D capabilities include geospatial referencing, humanoid animation, shared-state distribution using the IEEE Distributed Interactive Simulation (DIS) protocol, building prototypes, server-produced custom terrain, image overlay of photographic, cartographic or pseudocolor images, integrated physics for entity motion and sensor projection, a variety of user-interaction techniques, and scalable loading/unloading of interrelated scenes.

The multiple challenges involved in modeling deformable surfaces cannot each be solved in isolation. 3D graphics, underlying model representations and networked distribution at first appear to be different topics. Nevertheless, solutions for each area must simultaneously consider constraints and capabilities of other areas. Compatible integration within a Web-based framework allows effective use of deformable surfaces for diverse military simulations. This report presents results emphasizing standards-based interoperability, scalable architectures, demonstrated examples and directions for future work.

Figure 3. Abstract from *Web-Based 3D Graphics Rendering of Dynamic Deformation Structures in Large-Scale Distributed Simulations* provides a detailed reference showing the value of real-time web-based 3D graphics. [Brutzman 03]

E. LARGE-SCALE VIRTUAL ENVIRONMENTS (LSVES)

Large-scale virtual environments (LSVEs) can link hundreds of people and artificial agents with interactive 3D graphics, massive terrain databases, global hypermedia and scientific datasets. To do this, four key communication methods are used: lightweight messages, network pointers, heavyweight objects and real-time streams. For each of these four methods, bandwidth and latency must be carefully considered. [Brutzman 04]

Large-scale virtual environments and XTC face similar challenges and solutions to widely distributed real-time streaming. Specifically:

- Multicast protocols permitting moderately large real-time bandwidths to be efficiently shared by an unconstrained number of hosts.
- MBone connectivity permitting live distribution of graphics, video, audio, DIS and other streams worldwide in real time. [Brutzman 04]

The article "Exploiting Reality with Multicast Groups" [Macedonia 95] describes groundbreaking research on increasing the number of active entities within a virtual environment by several orders of magnitude. Multicast addressing and the DIS protocol are used to logically partition network traffic according to spatial, temporal, and functional-entity classes. "Exploiting Reality" further explains virtual environment network concepts and includes experimental results. This work has fundamentally changed the distributed-simulation community, showing that very large numbers of live and simulated networked players in real-world exercises are feasible. The military origins of this problem will become irrelevant as our abilities to effectively interact socially, scientifically, and educationally scale to the hundreds of thousands. [Brutzman 04]

In practice, widespread multicasting for LSVEs has been difficult to achieve due to administrative, routing and social barriers. The Jabber protocols hold great promise as a potential mechanism to achieve these long-sought breakthroughs.

F. EXTENSIBLE MODELING AND SIMULATION FRAMEWORK (XMSF)

The Extensible Modeling and Simulation Framework (XMSF) can be found at http://www.movesinstitute.org/xmsf/xmsf.html. XMSF is defined in Figure 4:

> A set of Web-based technologies, applied within an extensible framework, that enables a new generation of modeling and simulation applications to emerge, develop, and interoperate. Specific subject areas for XMSF include Web/XML, internet/networking, and modeling and simulation (M&S). The precepts of XMSF are:
>
> • Web-based technologies applied within an extensible framework will enable a new generation of M&S applications to emerge, develop and interoperate
>
> • An extensible framework of XML-based languages can provide a bridge between forthcoming M&S requirements and open/commercial Web standards, while continuing to support existing M&S technologies
>
> • Compatible and complementary technical approaches are now possible for model definition, simulation execution, network-based education, network scalability, and 2D/3D graphics views.
>
> • Web approaches for technology, software tools, content production and broad use provide best business cases from an enterprise-wide (i.e. worldwide) perspective.

Figure 4. Extensible modeling and simulation framework defined [Brutzman, et al. 2003]

The goal of XTC is to fit XML-based chat within the capabilities of XMSF. Multiple demonstration examples are now underway to test capabilities and achieve diverse cross-language interoperability via chat transport of web services.

G. DISTRIBUTED INTERACTIVE SIMULATION (DIS) PROTOCOL DIS-JAVA-VRML PROJECT

The IEEE Distributed Interactive Simulation (DIS) Protocol is used to create a common, interactive, virtual interface within a multi-system network by communicating state information. DIS provides the common interfaces, architectures and structures required to support this integrated environment. The goals of the DIS-Java-VRML project (http://www.web3d.org/WorkingGroups/vrtp/dis-java-vrml) are as follows:

- Complete a freely available reference Java implementation of the DIS protocol
- Produce a set of recommended practices for mapping between DIS and VRML worlds
- Develop assorted DIS utilities: record/playback, DIS viewers, etc.
- Develop math libraries and physics libraries

The DIS-Java-VRML project provides a valuable resource for testing example application interactions between XTC and DIS.

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

The C2IEDM is a NATO initiative to standardize the information exchange between coalition partners by creating a standard data structure. The C2IEDM is based on the Battlespace Generic Hub (BGH) concept which applies broad categories to military activities within the battlespace. [Neushul 03] XTC can use the BGH as a guide to define the XML tags so that it can interface with coalition systems. Successfully implementing C21Edm capabilities in XTC applications is expected to provide internationalized coaliton-capable tactical communications with consistent context and semantics.

I. XML BINARY SERIALIZATION USING CROSS-FORMAT SCHEMA PROTOCOL (XFSP)

XFSP is of particular interest to XTC because of its ability to compress message streams and its potential for higher performance in a low or noisy bandwidth environment. XFSP is described as follows:

> XFSP has been developed as a general approach to binary serialization of XML documents. Elements and attributes are replaced via a tokenization scheme which carefully preserves valid XML document structure. XFSP uses XML schema as the basis for determining key document parameters such as legal elements, attributes and data types. Originally motivated by the flexible definition of networking protocols, binary serialization of XML via XFSP appears suitable both for message streams and document-storage streams.

> An open-source XFSP software implementation written in Java demonstrates the viability of the XFSP approach for XML document serialization and deserialization. Compression measurements using Virtual Reality Modeling Language (VRML 97) scenes have shown expected compression results:
>
> • XML encoding of non-XML data (e.g. translation of VRML .wrl scenes into .x3d documents) typically increases file size.
>
> • XML-aware compression of XML encoded data (in this case via XFSP) provides superior compression to gzip compression of the original data. This occurs because the regularity of document structure is more exploitable than alphanumeric patterns.
>
> • Deserialization of compressed XML as in-memory data structures provides higher performance since a second parse of string-based data is not required. [Brutzman, McGregor 03]

Application-related communications will be performed as web-service requests. Binary XML compression of messages and data streams will show greater compression performance than is possible with gzip while retaining self-check schema-validation capabilities of XML. NPS further expects to allow optional addition of Hamming-code forward error correction (FEC) to all binary XML messaging, allowing receiver-side error detection and correction without retransmission. When combined with improved XML interoperability, such capabilities can be considered superior to the majority of existing machine-to- machine tactical communication encodings used by fleet systems.

J. SUMMARY

This chapter describes a wide range of topics that are relevant to XML-based tactical chat. Topics covered include multicast backbone (MBone), selective reliability multicast, X3D graphics, large-scale virtual environments, XMSF, the DIS protocol and DIS-Java-VRML project, and XFSP. Compression for XML-based messaging.

THIS PAGE INTENTIONALLY LEFT BLANK

III. TACTICAL, TECHNICAL, AND ADMINISTRATIVE CHAT REQUIREMENTS

A. INTRODUCTION

This chapter describes the tactical, technical, and administrative requirements of chat for the military environment. Tactical requirements define when and why chats are being conducted in a military environment, as well as the potential for increased benefits of chat. Technical requirements include enhancements that may be gained by leveraging the structure and flexibility of messaging and XML, as well as solutions to specific technical hurdles for military chat applications. Administrative requirements refer to rules concerning chat conduct, information assurance (IA) policies, and implementation at the user level that enable chat to proceed effectively.

B. GOAL OUTCOMES

The goal in defining tactical, technical, and administrative requirements is to better understand the uses of chat in a military environment so that end-user applications can flexibly support its many potential uses. Chat is already changing the way operations are conducted in the military, and better defining chat's requirements may improve (or indeed revolutionize) all tactical communications. A chat system can incorporate various human communications such as e-mail and message traffic, integrating these with system information to potentially reduce user information overload. The same chat system can handle system-to-system communications, greatly reducing the complexity of modern networking.

C. TACTICAL REQUIREMENTS

1. Overview

Recent doctrinal emphasis on network-centric warfare has led to unprecedented connectivity among operational forces. As a result, significant command-and-control functionality previously dependent on voice and message circuits has shifted to nontraditional resources such as e-mail, instant messaging, and chat. Chat in particular has become the tool of choice for real-time coordination and planning among distributed operational forces. In fact, operational units at all levels of command, from component and task-force commanders to individual ships, have assigned watchstanders to monitor relevant tactical chat rooms [Houlihan 03]. At present, operational chat utilization can be divided into three fairly broad categories: command and control, operational planning and execution, and administrative coordination. An additional advantage to the use of chat by operational forces is the ability to automatically maintain chat-room logs and archives, which provides a valuable resource to analysts responsible for after-action operational "lessons learned" and measures of effectiveness.

2. Command and Control

Reliable digital connectivity between watchstanders deployed in dispersed military units enables operational commanders to maintain real-time contact with subordinate units, regardless of physical location. Tactical chat provides an interactive, real-time means of passing information to all levels of the chain of command simultaneously. In particular, recent operations have shown that situation reports, execution checklist milestones, and casualty reports, as well as other draft reports, are now often passed via chat. This occurs with a level of detail not practical in voice circuit communications and a timeliness not available in record-message traffic. Additionally, instantaneous feedback available to all chat participants decreases potential confusion and miscommunication, facilitating dissemination of amplifying and clarifying information.

The following excerpt from a recent article provides a good example of chat being used for real-time tactical coordination in Operation Iraqi Freedom as described by RADM Paul F. Sullivan, USN:

"They [the sub crews] were sitting there with four different chat rooms, real-time, all interconnected 24/7 with Command Authority [and with other subs]. When one sub had a firing problem, another helped with troubleshooting and getting the missile ready to launch" [Sullivan 03].

3. Operational Planning and Execution

One of the greatest operational benefits of chat is its applicability to operational planning and execution. Chat has quickly become a principal means of communicating intentions and requirements between dispersed units collaborating in operational or training events. The ability to bring all participants together for planning on short notice with no logistical or operational overhead and to provide interoperability among dissimilar units, facilitates the flexibility to adapt plans on the fly in a dynamic environment. Event scheduling and coordination, deliberate planning of all aspects of future events (including requests for information, phased campaign milestones, supported/supporting unit coordination, strike planning, troop movements, logistics, and peer to peer planning of specifics), and real-time planning in response to short-fused operational exigencies such as the need for search and rescue and personnel recovery, weather degradation, and material casualties, have all benefited from the real-time connectivity provided by chat. Further, this planning capability applies to and has proven equally valuable at many levels of command, from component commanders planning macro aspects of major campaigns to individual units planning discrete evolutions. During the Seventh Fleet's fleet-battle experiment "Kilo," chat was used extensively to pass information and coordinate among coalition forces. [Schacher 03]

4. Coordination

Perhaps superficially less compelling, but no less important to operational success, is the employment of chat to facilitate administrative and logistical coordination among units. Connectivity between parent and detached units, superior and subordinate units, and supporting and supported units has proven an invaluable resource for coordinating administrative support, logistics, technical support, and a plethora of other mundane requirements. While voice circuits and message traffic provide only cumbersome, often unreliable means of coordinating needs, chat provides robust person-to-person real-time connectivity that has been proven to provide inestimable support time and time again.

A good example of the warfighting advantages of chat utilization is the NUWC chat implementation for managing Tomahawk land-attack missile (TLAM) capability among deployed submarine forces. By monitoring various tactical reports, TLAM inventory reports, and casualty reports, and providing technical guidance and advice to deployed units, NUWC was able to support both firing units and TLAM strike coordinators to ensure that submarine forces supporting contingency operations in Iraq were operationally effective. [McCormack 03] This article is attached with permission from NUWC as Appendix D.

5. Operational Analysis

Logging of chat-room traffic provides an excellent resource for reconstruction and analysis of operational events. Of particular interest, chat logs provide insight into decision-making processes unavailable by other means. Two shortcomings of the operational use of traditional chat in this regard are the use of unlogged private chat rooms (wherein decision makers discuss a situation before promulgating decisions in the logged room) and the inherent difficulty of gleaning relevant information from reams of freeform conversation. While overuse of private rooms remains a possibility (but one that can be dealt with), the highly structured nature of schema-governed XML is able to trivialize the data mining of chat logs for a variety of reconstruction purposes. Chat logging is not well supported and needs to be improved. Chat logging is primarily needed on servers. Augmenting server logs with additional logs on individual clients may further facilitate after-action correlation and analysis. This will be especially helpful in determining overall system reliability and distributed actions during periods of intermittent connectivity.

6. Shortcomings of Current Chat Systems

While present commercial chat functionality is sufficient to have raised its status to valued operational tool, its freeform nature and proprietary software constraints introduce two significant limitations. While it has been noted that the freeform discourse of chat systems makes data analysis difficult, this is in fact simply one manifestation of a larger shortcoming. Because of the problems in parsing free-form chat text, it is difficult to provide for automated data collection, collation, and usage in new capabilities such as tactical updates, post-mission operational analysis, and watch turnover. Commercial software executables provide zero confidence that information assurance requirements are met—and surreptitious eavesdropping or relaying of tactical messages is intolerable. Further, proprietary systems make the problem of application maintenance and adaptation to military requirements all but intractable. The bottom line is that proprietary protocols and applications benefit the vendor far more than the end user. A customized XTC client is needed to capitalize on the many potential benefits of chat.

7. Recommended Tactical Requirements

Figure 5 presents the recommended tactical requirements for chat.

- 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 an open-source solution that provides information assurance and is extensible for future requirements.

Figure 5. Recommended tactical requirements for XML-based tactical chat.

D. TECHNICAL REQUIREMENTS

1. Overview

The technical requirements for tactical chat comprise two major concerns: data formats and application design. Data format issues include XML markup, XSLT, BGH compatibility, MIME types, URLs, and enhanced data-mining functionality. Application-design concerns include asynchronous data exchange, thin-client configuration, interoperability, firewall-policy compatibility, and open-source software design.

2. XML Markup

XML is a standard for marking up data, used to create a language that describes that data [Hunter 01]. XML describes many types of data, as seen in the related work chapter, and can be applied to describe chat data. By describing this data with XML, a user can provide forms for submitting standard reports, as well as for storing and sorting data for customized outputs. Functionality of this sophistication can promote chat from an arbitrary discussion environment to a formalized reporting tool.

While overall chat utility can improve significantly through XML, it is important that no current chat-created functionality be sacrificed. It is unlikely that all tactical information exchanges will map cleanly into a highly structured (and finite) set of message types. At least one type of XML-based plain-prose message must be available to handle these types of conversations. Further, while their use should be limited, offline conversations (i.e., those unlogged and unseen by other chatters) have proven popular, and seem to enhance collaboration to some degree. Thus payload flexibility is essential.

3. Multiple Message Types

The system will be able to process plain prose, message text format (MTF), BGH Tactical Markup Language (BGHTML), HTML, and BGH ATTCS reference model messages. Additionally, use of extensible stylesheet language transformations (XSLT) and schema-governed message templates will allow for arbitrary addition, deletion, and modification of messages.

4. Message Validation and Data Integrity

All messages will be validated against the schema by the server. This will enforce data integrity throughout the system by rejecting invalid messages. Additionally, the use of schema-governed message templates and XSLT to automatically generate HTML pages for message composition and submission will provide an intuitive method for users to generate messages and compose valid messages.

5. BGH Compatibility

Message fields that correspond to BGH elements will be compatible with the BGH architecture [CC2IEDM 2004]. XSLT will be used as required to map tactical chat messages to the BGH schema for automatic update of the BGH database. Since the BGH is essentially a theater-wide tactical data repository and clearinghouse, BGH compatibility provides compatibility with any other systems that rely on the BGH, such as the Global Command and Control System (GCCS) and the Command and Control PC system (C2PC).

6. MIME Types

Mime types are used to describe e-mail. By integrating MIME types into a tactical chat application, the need for a separate e-mail system can be reduced or eliminated. The e-mail can be XML-ized and filtered for a customized interface.

7. URLs

As described at http://www.w3.org/Addressing:

> Uniform Resource Identifiers (URIs, aka URLs) are short strings that identify resources in the web: documents, images, downloadable files, services, electronic mailboxes, and other resources. They make resources available under a variety of naming schemes and access methods such as HTTP, FTP, and Internet mail addressable in the same simple way. They reduce the tedium of "log in to this server, then issue this magic command ..." down to a single click.

The ability to pass URLs is a necessary element of any Web application, including chat. Different URL spaces will correspond to different access requirements and classification levels of various tactical networks.

8. Cell Phone SMS

Short-message system (SMS) technology enables short messages to be sent between cellular phones. As the capabilities and presence of cell phones continues to grow it is important to be able to communicate from a chat application to the mobile user, and vice versa.

9. Data Mining

Tag sets will be designed to provide for easily implemented data mining of the chat log. Well-defined tag sets will aid XSLT and XPath in culling relevant elements from the log for operational reconstruction and analysis and might automatically generate any imaginable summary report (e.g., watch turnover summaries, logistics summaries, schedule requests, etc.).

10. Non-Text Inclusion

The Jabber protocol allows for non-text content as a future enhancement. This capability may become practical and useful in encoding and transporting audio, video, whiteboard, and other non-text data that is routinely used in planning. While not envisioned as a replacement for current video teleconferencing capabilities, the asynchronous nature of the Jabber protocol, coupled with the structured data of schema-governed XML (not to mention the decreased bandwidth and technical overhead requirements) may make this an attractive option in many circumstances.

11. Asynchronous

The military uses chat in an environment where connectivity is by no means guaranteed. Because of this, the protocol for tactical chat must be asynchronous, meaning it cannot crash or lock the system if the chat cannot be completed immediately. When necessary, the system must use a store-and-forward approach to messaging, i.e., storing messages on the local server and then forwarding or synchronizing when the restored connection allows.

12. Launch Synchronous Streams

Many tools that can enhance the chat experience use synchronous protocols to communicate. Because tactical chat uses an asynchronous protocol it is necessary that it be able to launch synchronous streams. One such tool is the Network Education Ware (NEW) project which enables whiteboarding, video and audio features in real time. More information can be found at http://netlab.gmu.edu/NEW/index.shtml.

13. Thin Client

A "thin client" for chat means that no applications need be loaded onto the client, and the user can conduct a chat session via a Web browser. A thin client is desirable because it requires no installation and is easily maintained at a central server location. In addition, the current development requirements for NMCI state that applications need to be Web enabled, consisting of a server-side component and a Web-based interface for the client [TFWeb 03].

14. Interoperable

Joint- and inter-service forces have used a myriad of chat programs that do not interoperate and are proprietary and expensive. The current decision-support systems IWS, Groove, and DCTS are not interoperable and have forced the JFC to look to standards-based systems as a replacement [Boyd 03]. For ease of joint and multinational use, tactical chat must implement a standardized system that can work with other standardized systems.

15. Firewall Policy Compatible

Because of strict DoD firewall policies, many chat programs are blocked by firewalls. Tactical chat needs to run across an approved port for server-to-server (S2S) connections.

16. Open Source - Flexible and Extensible

“Open source” means that code is in the public domain, using an open standard. It enables thorough code checking and constitutes a nonproprietary solution that is both flexible and extensible.

17. Recommended Technical Requirements

Figure 6 presents the recommended technical requirements for chat:

- 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 6. Recommended technical requirements for XML-based tactical chat.

E. ADMINISTRATIVE/SOCIAL REQUIREMENTS

1. Overview

Getting value from a chat session is more than using the technology successfully. For leaders, it requires the ability to control the "who, what, when, where and why" when chat communications take place. Tools that permit scheduling, role definitions, and large-scale discussions can be built into a tactical chat system. Additionally, chat must meet specific IA requirements to be implemented in the DoD. [DODI 5200.40]

2. Managing Large Discussions

Broad discussions provide unique challenges that have been researched and defined. Chat is a form of a large-scale discussion and will therefore benefit from applying the best ideas of large-group management. Some of these are frontloading, negotiation, action planning, and follow up. Below are some highlighted concepts from virtualTeamworks.com that assist large group discussions in a virtual environment:

- **FRONTLOADING** is asking questions before (instead of after) the learning experience, in order to begin (rather than end) with reflection, and so that behavioral changes can occur during the most powerful part of the experiences.
- **INTERVENING** is interrupting a team during their experience by taking "timeout" and asking questions that get them back on track. However, it is reserved for times when the team may not notice their efforts have stalled and when interrupting will not interfere with their progress or create facilitator dependence.
- **SOLUTION-FOCUSED QUESTIONING** centers on what people are already doing well (not doing poorly) and exceptions to the problem (not why the problem happens). Therefore, it works best with resistant teams in trouble that may no longer be willing to explore their difficulties.
- **CLARIFICATIONS** are used with people or teams that are simply uninformed. Clarifying information should come from the people or teams and not the facilitator.
- **NEGOTIATION** is used with people or teams that disagree. It proves useful to discover what they really want and should be conducted in a respectful manner with their best interests at heart.
- **REFRAMING** is used with people or teams that are consciously opposing change, and causes them to reconsider their behaviors, from fresh perspectives by reviewing the content and context of what success will really mean to them (e.g., the "confusion technique" where the electronic facilitator acts very confused).
- **ACTION PLANNING** is the detailed preparation of strategies to create change and is where teams discuss and record: what they will do, why they will do it, where it will be done, when it will be completed, who will confirm the change, and how they will measure the difference. Typically, clients sign and share their plans in public.
- **FOLLOW UP** is conducted after the return back to work. This can take on many forms ranging from completing a work-based project, through coaching support communities, to further training and simulation events.

Figure 7. Concepts that promote large-group discussion in a virtual environment from http://virtualTeamworks.com [Virtual03]

3. Chat Room Administration and Conduct

The ability to effectively govern chat-room permissions and conduct is an important aspect of tactical-chat architecture. For instance, defining and implementing the roles of chat-room administrator, moderator, contributor, listener, and participant are required to ensure effective sessions. Additionally, the ability to define room permissions, delineating who can participate in a given room and to what extent (post, monitor only, no participation, etc.) must be defined. Finally, rules governing post content to ensure relevance and propriety will be implemented.

4. Scheduling Chat Sessions

The ability to schedule chat sessions, either in advance or on the fly, and to include participants automatically is an improvement that will aid both deliberate and crisis-provoked planning. Both regular sessions to support fixed timelines and ad-hoc meetings in response to emergent requirements can be implemented in the same manner, through a dynamic buddy list based on areas of interest. Upon login, users will indicate or be assigned areas of interest (joining a buddy list of others with the same interests) based on their responsibilities. When sessions pertaining to a specific interest are scheduled, all users with that interest will be notified, or possibly brought into the chat automatically.

5. Information Assurance

Information assurance is the "measures that protect and defend information and information systems by ensuring their availability, integrity, authentication, confidentiality, and non-repudiation. This includes providing for restoration of information systems by incorporating protection, detection, and reaction capabilities" [DODD 8501]. It is imperative that tactical chat meet the DoD requirements for IA as a prerequisite for its implementation.

To achieve DoD approval, tactical chat must attain a System Security Authorization Agreement (SSAA) as outlined in DoD Instruction 5200.40, *DoD Information Technology Security Certification and Accreditation Process* (DITSCAP) [DODI 5200.40]. An alternate avenue is to have tactical chat as an approved application in the NMCI and/or TFW framework instead of a standalone system. Further work is needed to ensure that a) chat communications and b) chat applications meet these requirements.

6. Recommended Administrative Requirements

Figure 8 presents the recommended administrative requirements for chat:

- 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 8. Recommended administrative requirements for XML-based tactical chat.

F. SUMMARY

This chapter defines the tactical, technical, and administrative requirements in developing a tactical chat application. The tactical requirements are to support command and control, operations, and coordination efforts, along with XML-izing and storing the chat data. Technical requirements are to broadly format data using XML while supporting various data formats under an open-source, asynchronous, interoperable environment. Administrative requirements are to define roles in military chat and enable proper scheduling and transmission of information.

THIS PAGE INTENTIONALLY LEFT BLANK

IV. OVERVIEW OF JABBER CHAT PROTOCOLS AND SOFTWARE

A. INTRODUCTION

This chapter explains what Jabber is and how it works, giving a brief history and recounting the benefits of the open-source, extensible, asynchronous, secure properties inherent to Jabber. The roles of clients and servers in a Jabber environment, the XML data format, Jabber IDs, the XML connection stream, and the attributes of that stream are all explained. Finally, examples of successful and failed Jabber conversations are provided.

B. WHAT IS CHAT?

Chat is "real-time communication between two users via a computer" [Webopedia]. Chatting can be conducted between two individuals, called instant messaging (IM), or a group of users in a chat room, called conferencing. Traditionally, chat consists of text based communications only. However, this report uses a broader definition of chat, in which any data on a computer has the potential to be sent as a real-time communication.

C. WHAT IS JABBER?

Jabber is an open-source, XML-based protocol for the real-time exchange of messages and presence between any two points on the Internet. The Jabber protocol is based on the Internet Engineering Task Force (IETF) -supported XMPP. The first implementation of Jabber technology is an IM network application that offers functionality similar to legacy IM systems such as IRC, AIM, ICQ, MSN, and Yahoo.

Jabber was started by Jeremie Miller in early 1998, and by late 1999 most of the Jabber protocol was developed, evolving along with the jabberd server and early clients such as Winjab (since deprecated in favor of Exodus) and Gabber. Jabbered 1.0 was released in May 2000, and in August of 2001, Jabber was placed under the auspices of the Jabber Software Foundation. As of 4 June 2003, IETF is looking at adapting the Jabber protocol as an IETF-approved instant messaging and presence technology.

The November 2003 issue of *ACM Queue* magazine is devoted to IM and chat. The article entitled "Nine IM Accounts and Counting" highlights the need to standardize chat protocols and discusses how Jabber is taking the lead in implementing XMPP. "XMPP is the more complete standard in that its work with the IETF is nearly finished, and it has been field proven over the course of almost five years with nearly 10 million users (actually, many more than that if you consider that the open-source community uses XMPP to communicate with MSN, ICQ, AOL, and Yahoo users).

Some of the standards bodies have made the interoperability problem too complex by trying to solve a rigorous, academically complete set of functionality instead of just defining an extensible core that can be added to over time. This approach leads to difficulties in incremental implementation and deployment. Jabber took the extensible approach with XMPP, so we have been able to bite off smaller stanzas and solve them independently." [ACM 03]

D. STATISTICS ON CURRENT CHAT USE

The value of chat is evident by the amount of commercial use it gets in the corporate world. AOL reports that more than a billion instant messages a day traveled across its AIM network by the end of the first half of 2003. IM increased by more than 25 percent since the end of 2002, and the amount of AIM network use solely by the corporate sector likely has doubled.

The search service Yahoo reports levels for the end of June 2003 reaching almost 700 million per day—more than 23 percent higher than at year-end 2002. IBM Lotus, as a provider of IM solutions targeting the business marketplace, reports an increase of more than a million corporate users from December 2002 to June 2003. The current corporate user base now totals more than 9 million. [Ritter03]

E. INTERNET RELAY CHAT (IRC)

Internet Relay Chat (IRC) was created in 1988 by the Finish programmer Jarkko Oikarinen and was the first multi-user chat program. The original goal of IRC was to enable the functionality of a computer based bulletin board system (BBS) in real-time. When it was discovered traditional BBS functionality didn't add anything to the multi-user chat, it was soon abandoned and IRC was created. [Oikarinen 00]

IRC is based on a client-server and server-server model. An IRC client program communicates with an IRC server, which is connected via the internet to other IRC servers, together making up an IRC network. The latest IRC protocol document is the Request for Comments (RFC) 1459 written in 1993; it is kept by the Internet Engineering Task Force (IETF) and can be found at http://ietf.org/rfc/rfc1459.txt?number=1459.

IRC is currently the most widely used chat protocol in the Navy. It is used by many clients including the proprietary chat client Microsoft's MS Chat which is included in the standard Information Technology 21 (IT-21) software load. The advantage of IRC over other chat clients is that it uses less bandwidth than competing proprietary clients such as Microsoft's Netmeeting. [Jara 03]

Some shortfalls of IRC include lack of support for the functions of logging, timestamps, and authentication that are required in the military environment. [Thomas 03] Additionally, current implementations of IRC in the military are proprietary and not approved by the Joint Staff Collaboration Working Group, a problem which will prevent its use on DoD networks as of October 2003. [Jara 03] Interestingly, Jabber allows for IRC bridges to provide connections to an IRC network if required.

F. ADVANTAGES OF JABBER

The four main advantages of Jabber are that it is open source, extensible, asynchronous, and secure. Because Jabber includes open-source implementations, it is free of charge and has the advantages of an open-source project, which include reliability and security [Wheeler 03]. The extensible characteristics of Jabber allow the user to harness the power of XML namespaces by extending the Jabber protocol for custom functionality. To increase interoperability, common extensions are managed by the Jabber Software Foundation (JSF).

The Jabber protocol is asynchronous, meaning that messages can be stored at the local server and forwarded once connected. Jabber provides security by allowing servers to be isolated from the public Jabber network. Many server implementations use secure-socket layers (SSL) for client-server communications, and numerous clients support Pretty Good Privacy/Gnu Privacy Guard (PGP/GPG) for end-to-end encryption; more robust security using simple authentication security layer (SASL) and session keys is under development.

G. JABBER CLIENT/SERVER RELATIONSHIP

Jabber is not a direct peer-to-peer (P2P) architecture, but rather is client-server oriented. The default for a Jabber client is to connect to a Jabber server on a TCP socket over port 5222. This connection is always on for the life of the client's session on the server. When sending a message, it goes from the sender to the sender’s server. If connected to the network, the sender’s server sends the message to the recipient’s server, which forwards it to the recipient. If the recipient is unavailable or the requested Jabber server cannot be found, the message is stored and later forwarded when possible.

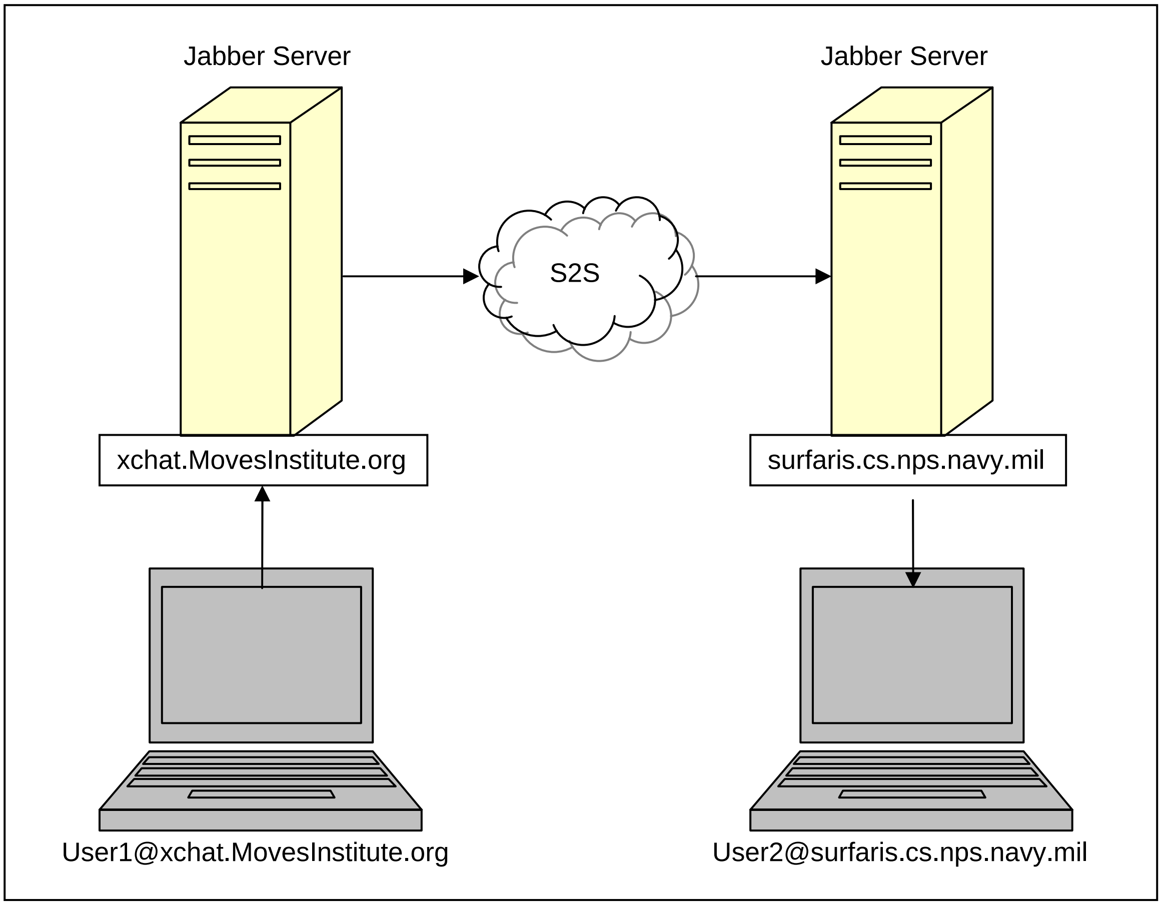


Figure 9. A diagram showing the flow of a chat message from a client to a server and then to the intended recipient.

Any domain can run a Jabber server. Each server functions independently of the others and maintains its unique user list. If server-to-server communications are enabled, a Jabber server can talk to any other Jabber server that is accessible via its network. A particular user is associated with a specific server, either through registration with a service provider or administrative setup within an enterprise.

A Jabber client is simple. It communicates with a Jabber server over TCP sockets and parses and interprets well-formed XML document fragments (sometimes call "Jabber stanzas") over that XML stream. A Jabber client understands the core Jabber data types: message, presence, and iq. In practice, many of the low-level functions of the client (e.g., parsing XML and understanding the core Jabber data types) are handled by Jabber client libraries, enabling client developers to focus on the user interface.

H. XML DATA FORMATS

XML is an integral part of the Jabber architecture because it is of utmost importance that the architecture be fundamentally extensible and able to express almost any structured data. When a client connects to a server, it opens a one-way XML stream from client to server, and the server responds with a one-way XML stream from server to client. Each session involves two XML streams. All communication between the client and server happens over these streams, in the form of small snippets (sometimes referred to as stanzas) of XML. The following shows a single XML stream fragment sent by a Jabber client:

```
<message from='user1@xchat.movesinstitute.org'
     to='user2@surfaris.cs.nps.navy.mil'>
   <body>This is a test message.</body>
</message>
```

Figure 10. A sample XML stream

Jabber's XML format can also be extended through official XML namespaces, through a process managed by the JSF. Custom namespaces for specialized applications can be created without any required standardization process or coordination.

I. JABBER IDS

A Jabber ID is easy to remember and uniquely identifies individual objects or entities for communicating instant messages and presence information. Jabber addresses are of the same form as e-mail addresses and are flexible enough to include other IM and presence schemes. Each Jabber ID (JID) contains a set of ordered elements. The JIDs are formed of a domain, node, and resource in the following format:

[node@]domain[/resource]
brutzman@xchat.MovesInstitute.org
dadevos@surfaris.cs.nps.navy.mil

Figure 11. Examples of Jabber IDs.

The domain name is the primary identifier. It represents the Jabber server to which the entity connects. Every usable Jabber domain should resolve to a fully qualified domain name. The node is the secondary identifier, representing the user. All nodes are found within a specific domain. However, the node is optional, and a specific domain (e.g., <conference.jabber.org>) is a valid Jabber ID. The resource is an optional third identifier. All resources belong to a node. Within Jabber, the resource is used to identify specific objects that belong to a user, such as devices or locations. Resources enable a single user to maintain several simultaneous connections to the same Jabber server; examples might be <brutzman@xchat.MovesInstitute.org/Exodus> vs. <brutzman@xchat.MovesInstitute.org/Rhymbox>.

J. XML CONNECTION STREAM

On connecting to a host, a node initiates an XML stream by sending a properly namespaced <stream:stream> tag, and the host replies with a second XML stream back to the node. Within the context of an XML stream, a sender is able to route a discrete semantic unit of structured information to any recipient. This unit of structured information is a well-balanced XML stanza, such as a message, presence, or IQ stanza. These stanzas exist at the direct child level (depth=1) of the root stream element. The start of any XML stanza is unambiguously denoted by the element start tag at depth=1 (e.g., <presence>) and the end of any XML stanza is unambiguously denoted by the corresponding close tag at depth=1 (e.g., </presence>). Each XML stanza may contain child elements or CDATA sections as necessary, to convey the desired information from sender to recipient. The session is closed at the node's request by sending a closing </stream:stream> tag to the host.

Thus a node's session with a host can be seen as two open-ended XML documents that are built up through the accumulation of the XML stanzas that are sent over the course of the session. In essence, an XML stream acts as an envelope for all the XML stanzas sent during a session.

```
-------------------
open stream
-------------------
<message to=''>
   <body/>
</message>
-------------------
<presence to=''>
   <show/>
</presence>
------------------
<iq to=''>
   <query/>
</iq>
-------------------
close stream
-------------------
```

Figure 12. Example fragment from a Jabber XML stream.

XML streams function as containers for any XML stanzas sent asynchronously between network endpoints. XML streams are used for the following types of communication: node to host "jabber:client"; host to host "jabber:server"; and service to host. In a service-to-host communication, the service initiates communications to the host with “jabber:component:accept,” and the host initiates communications to the service with “jabber:component:connect.” These usages are differentiated through the inclusion of a namespace declaration in the stream from the initiating entity, which is mirrored in the reply from the receiving entity.

XML streams are used to transport a subset of XML and have specific restrictions. XML streams should not contain processing instructions, undefined entities, comments, or DTDs. Any now-XML data inside the stream is ignored.

K. ATTRIBUTES OF THE JABBER STREAM ELEMENT

Stream elements contain three elements: `to`, `from`, and `id`. The `to` element is from the initiating entity to the receiving entity, and is set to the JID of the receiving entity. The `from` element is sent by the receiving entity to initiating entity, and is set to the JID of the receiving entity, granting access to the initiating entity. The `id` element is sent by the receiving entity to the initiating entity. It is a unique identifier created by the receiving entity to function as a session key for the session. The following table shows what action is taken when receiving the different stream elements:

Element	Initiating to Receiving	Receiving to Initiating
`to`	JID of Receiver	Ignored
`from`	Ignored	JID of Receiver
`id`	Ignored	Session Key

Figure 13. Correct responses to Jabber stream elements

The 'xmlns' namespace declaration is required and is used in both XML streams in order to scope the allowable first-level children of the stream element for both streams. This namespace declaration must be the same. The 'xmlns:stream' namespace declaration is also required in both XML streams. The value must be "http://etherx.jabber.org/streams."

L. JABBER STREAM ERRORS

The stream element may also contain <stream:error/> as a child element, signifying that a stream-level error has occurred. Errors may occur at the level of the stream. The following table lists possible stream errors:

- Sending of invalid XML
- Shutdown of a host
- Internal server error such as the shutdown of a session manager
- An attempt by a node to authenticate as the same resource that is currently connected.

Figure 14. Possible reasons for a stream error in a Jabber XML stream.

If an error occurs at the level of the stream, the entity (whether initiating or receiving) that detects the error should send a stream error to the other entity specifying why the streams are being closed, closing with a </stream:stream> tag.

```
<stream:stream ...>
   ...
   <stream:error> Error message (e.g., "Invalid XML") </stream:error>
</stream:stream>
```

Figure 15. Example of a stream-error closing stream.

M. JABBER EXAMPLES

The following two examples show a successful and a failed Jabber conversation:

```
NODE: <stream:stream
        to='host'
        xmlns='jabber:client'
        xmlns:stream='http://etherx.jabber.org/streams'>
HOST: <stream:stream
        from='host'
        id='id_123456789'
        xmlns='jabber:client'
        xmlns:stream='http://etherx.jabber.org/streams'>
NODE: <message from='node@host' to='receiving-ID'>
NODE:      <body>This is a test message. </body>
NODE: </message>
HOST: <message from='receiving-ID' to='node@host'>
HOST:      <body>I received the message. </body>
HOST: </message>
NODE: </stream:stream>
HOST: </stream:stream>
```

Figure 16. A successful Jabber conversation: Message sent received and acknowledged.

```
NODE: <stream:stream
        to='host'
        xmlns='jabber:client'
        xmlns:stream='http://etherx.jabber.org/streams'>
HOST: <stream:stream
         from='host'
         id='id_123456789'
         xmlns='jabber:client'
         xmlns:stream='http://etherx.jabber.org/streams'>
NODE: <message>lt;body>Bad XML, no closing body tag!</message>
HOST: <stream:error>Invalid XML</stream:error>
HOST: </stream:stream>
```

Figure 17. A failed Jabber conversation: Client post message was malformed, XML thus invalid, resulting in error response.

N. SUMMARY

Jabber is a recently developed, open-source, chat protocol that enables an asynchronous XML formatted conversation. It is reliable, secure, and can be extended and customized to meet tactical-chat functionality. This chapter describes the value of chat, XML, and Jabber. It also explains JIDs, and Jabber XML streams.

V. JABBER DEPLOYMENT GUIDE

A. INTRODUCTION

This chapter shows how to configure a Jabber client and server and send an XML message to a Jabber IM client. The first section includes the steps needed to install a client and an evaluation of three Jabber clients. The next section describes how to install a Jabber server, in particular describing how NPS installed the `jabberd` daemon on the hosts `surfaris.cs.nps.navy.mil` and `xchat.MovesInstitute.org`. The last section describes how to configure a Web server with Tomcat and Java servlets to support an HTTP post to a Jabber server.

B. JABBER CLIENT SET-UP

The following section gives step-by-step instructions on how to install and configure a Jabber client. Three clients are also evaluated: Rhymbox, Exodus, and BuddySpace.

1. Download a Client

The first step is to download a Jabber client. The following table lists Jabber clients tested for Windows as part of this report, and where they can be found:

BuddySpace	http://buddyspace.sourceforge.net
Exodus	http://www.jabberstudio.org/projects/exodus/releases
Rhymbox	http://www.rhymbox.com/download

Figure 18. List of tested Jabber clients and source locations.

Rhymbox and Exodus are used primarily in this report because they currently have the most refined interface and also include user-friendly XML-aware debug windows. Additional chat clients can be found at http://www.jabber.org.

2. Log onto a Jabber Server

The next step is to log onto a Jabber server. The user needs network access to a Jabber server in order to create a Jabber account. Currently there are two NPS servers, `surfaris.cs.nps.navy.mil` inside the firewall and `xchat.movesinstitute.org` outside. Clients cannot log in across the firewall boundary, requiring them to use a server on the same side. For example, a client inside the NPS firewall can only log into Jabber servers inside the firewall, and a client outside the firewall can only log into Jabber servers outside the firewall. However, Jabber servers can communicate with each other via an S2S port that has been opened in the firewall. This allows clients who have logged into a protected NPS Jabber server to send messages to clients logged into Jabber servers outside the firewall and vice versa. Because only plain-text XML messages can pass over this channel, virus threats are greatly reduced. There is also an experimental server for NPS users at `dickdale.cs.nps.navy.mil` and a server hosted by FNMOC at `jabber.metnet.navy.mil`.

In addition to the server name, the user needs to know the port number. The default for the Jabber client-to-server protocol is port 5222; however FNMOC's server uses port 8080. The default S2S port number is 5269. Below is a topology of available servers at NPS.

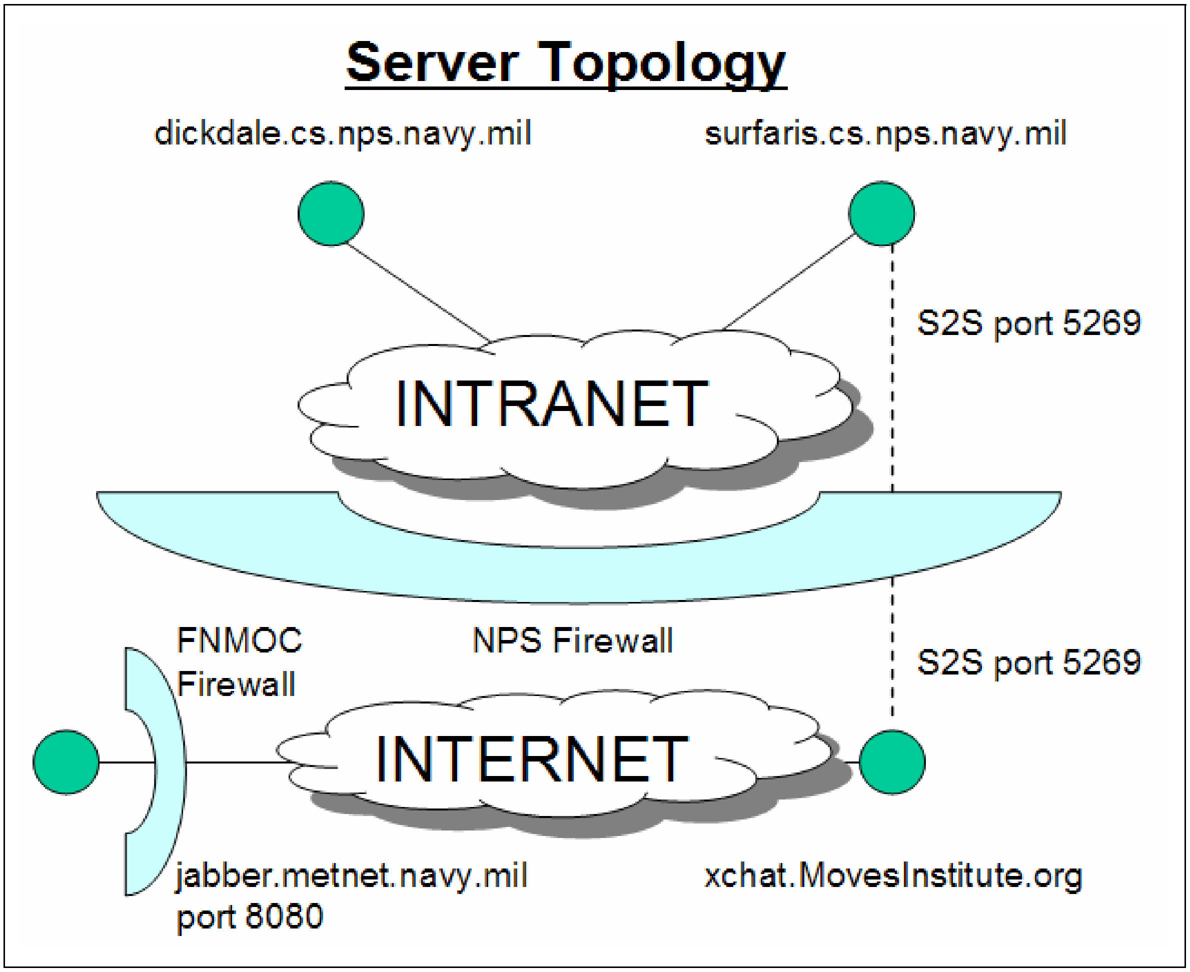


Figure 19. Diagram of the server topology inside and outside the firewall at NPS.

To set up a Jabber account inside the firewall, enter the following information (if outside the firewall substitute `surfaris.cs.nps.navy.mil` with `xchat.movesinstitute.org`):

Profile
Create a Username: (User choice)
Server: **surfaris.cs.nps.navy.mil**
Password: (User choice – must have one)

Connection
Type: **Normal**
Host: **surfaris.cs.nps.navy.mil** (Same as Server)
Port: **5222**

Figure 20. Settings for logging into a Jabber server

When logging into the server for the first time, users must agree to create a new account. Some clients give the option of filling out additional information on first login. Such setup can also be performed at a later time.

3. Add an Individual to a Buddy List

To add an individual to a buddy list, the user will need to know the full user name and server name. Then enter the person's JID: (user name)@(server name), which looks exactly like an e-mail address. Jabber will not add the user to a buddy list until it receives confirmation from the identified individual. For more information on JIDs, see the Jabber IDs section in Chapter IV.

4. Join a Conference Room

The next step is to join a conference room. The rooms have the same name and server location, whether joined from inside or outside the firewall. In order to join a conference room, enter the conference rooms JID into the Jabber client. The menu options for this process may differ slightly, depending on the client. For example in Rhymbox the click Find a Conference Room then Join Room, and then enter the room name `savage` or `xj3d`. The JIDs for rooms of interest are:

Chat Server	Conference Room	JID
XChat	Savage	`savage@conference.xchat.movesinstitute.org`
XChat	Xj3d	`xj3d@conference.xchat.movesinstitute.org`

Figure 21. Conference room JIDs of interest.

When joining a conference room a new JID will be created for the user's conference room session. The JID will be the [conference room]@[conference server JID]/[user's nickname]. Depending on the client, the user's nickname is set in the users profile or upon logging into the conference room.

5. Client Comparison

The different features of the clients are shown in Figure 22 below, excerpted from the Jabber clients support page at http://www.jabber.org/user/clientlist.php. Figure 23 explains what is meant by each of the client features categories below.

Client	**Basic Chat**	**Groupchat**	**Headline Support**	**Browser Support**	**x:data Support**	**Unicode Support**		**File Transfer**	**Message History**	**Invisible Support**	**Proxy Support**	**SSL Support**	**Sound Notifications**	**Emoticons**	**Skinning**	**Reply Indicator**
Exodus	★	★	★	★	★	⯪		★	★	★	★	★	★	★	☆	★
RhymBox	★	★	★	★	★	★		★	★	★	★	★	★	★	☆	★
BuddySpace	★	★	★	★	⯪	★		★	★	☆	★	☆	★	★	★	☆

Legend	**Support Level**
★	Full
⯪	Partial
☆	None

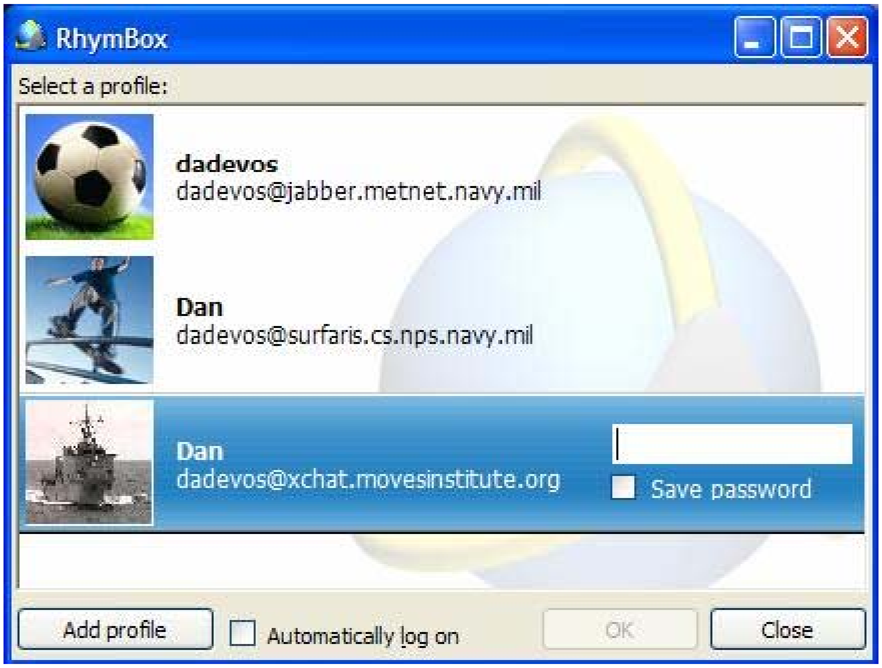
Figure 22. The features currently implemented in the Jabber clients used in this report. [Client 03]

Client Features	Description of Features
Basic chat (also known as IM)	Peer to peer chat.
Groupchat (superseded by **Multi-User Chat** and also known as conferencing)	Multiple users reading and chatting simultaneously.
Headline Support	Headlines consist of a URL and a subject.
Browser Support (replaced by **disco**)	Service Discovery http://www.jabber.org/jeps/jep-0030.html.
x:data Support	Generic data gathering/reporting in Jabber. URL: http://www.jabber.org/jeps/jep-0004.html.
Unicode Support	Allows multiple languages to be used.
File Transfer	Send files.
Message History	Creates a message history that can be saved.
Invisible Support	Track presence type.
Proxy Support	Allows the use of a proxy in connection settings.
SSL Support	Allows the use of secure socket layer (SSL) in connection settings.
Sound Notifications	Plays sound when message received
Emoticons	Replaces special key strokes with pictures
Skinning	Allows the user to change the look of the client with a "skin."
Reply Indicator	Message events such as "Is-Composing."

Figure 23. Description of the features that are currently implemented in Jabber clients.

6. Rhymbox

RhymBox is a Jabber client for chat with a Jabber server and can be found at http://www.rhymbox.com. RhymBox has a shared-source license and is written in C++. The first screen in Rhymbox is the login screen, which is easy to use and lists all user accounts. Icon images smaller than 8KB can be used to simplify user/role login selection.



Select a profile:

dadevos
dadevos@jabber.metnet.navy.mil

Dan
dadevos@surfaris.cs.nps.navy.mil

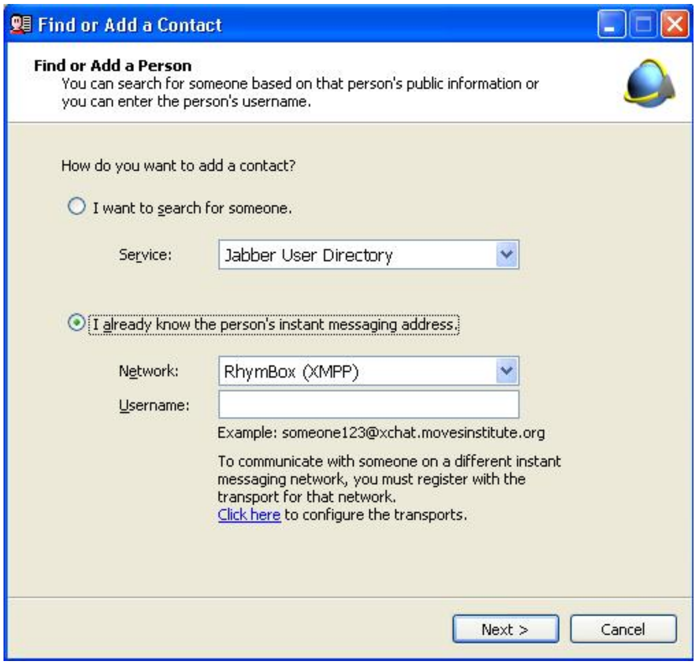
Dan
dadevos@xchat.movesinstitute.org

□ Save password

□ Automatically log on

Figure 24. Multiple profiles lets users have accounts on multiple servers or share the RhymBox application with others on the same computer.

The next screen, Add a Contact, allows the user to search a directory for a user's name or type the JID of a known contact in the username box.



Find or Add a Person

You can search for someone based on that person's public information or you can enter the person's username.

How do you want to add a contact?

○ I want to search for someone.

Service: Jabber User Directory

● I already know the person's instant messaging address.

Network: RhymBox (XMPP)

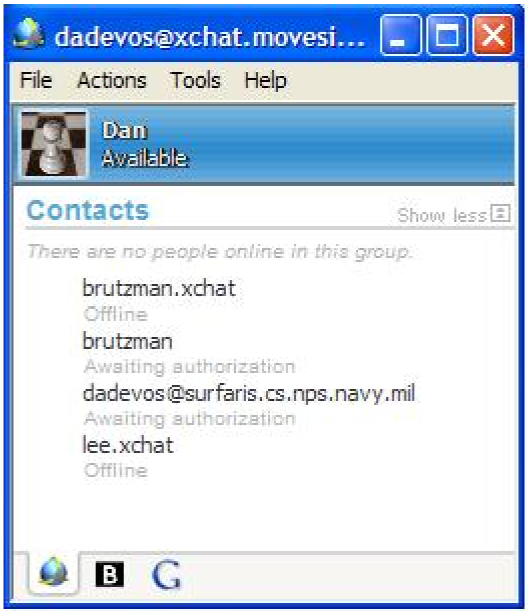
Username:

Example: someone123@xchat.movesinstitute.org

To communicate with someone on a different instant messaging network, you must register with the transport for that network.
Click here to configure the transports.

Figure 25. Screen shot showing how to add contacts with the RhymBox client.

The next screen shows the user's contacts and their on-line status. The picture next to the name is an avatar and can be customized from the Actions tab. Select Actions/Change My Avatar if the picture is not available, then the Custom Image button. Pick a picture of type gif, jpg or png that is less than 8KB.



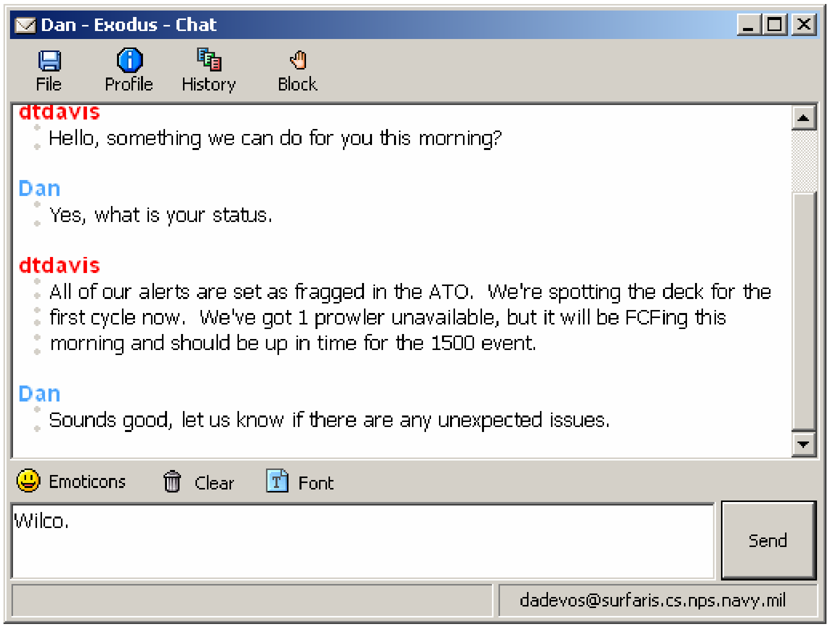
Dan
Available

There are no people online in this group.

brutzman.xchat
Offline
brutzman
Awaiting authorization
dadevos@surfaris.cs.nps.navy.mil
Awaiting authorization
lee.xchat
Offline

Figure 26. Above is a screen shot showing Jabber contacts' online status in RhymBox.

The next screen is the standard chat window for RhymBox.



dtdavis
Hello, something we can do for you this morning?

Dan
Yes, what is your status.

dtdavis
All of our alerts are set as fragged in the ATO. We're spotting the deck for the first cycle now. We've got 1 prowler unavailable, but it will be FCFing this morning and should be up in time for the 1500 event.

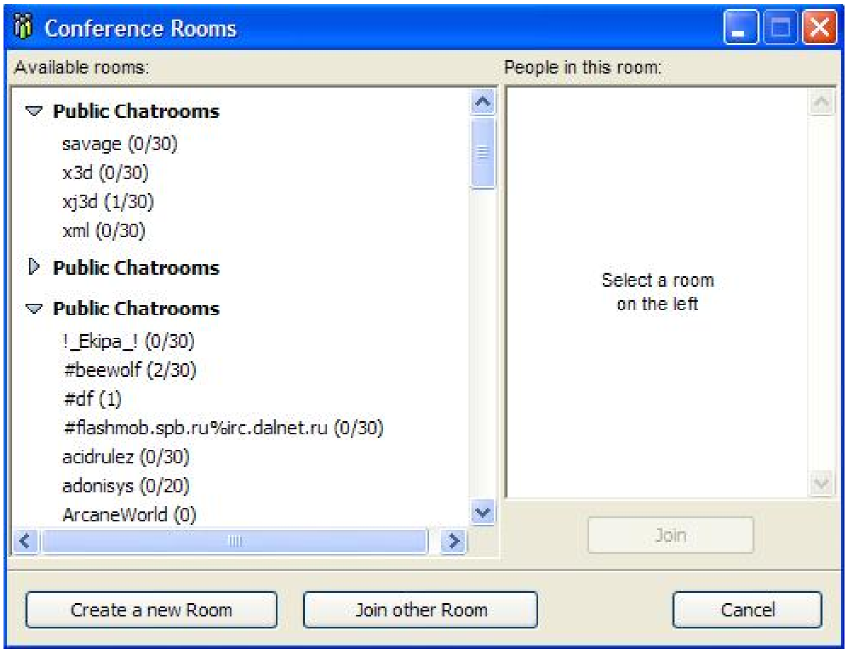
Dan
Sounds good, let us know if there are any unexpected issues.

Wilco.

dadevos@surfaris.cs.nps.navy.mil

Figure 27. Above is an example of a P2P chat session in RhymBox.

The next screen lists all the conference rooms available and gives the user the number of people in each room plus the option to create new rooms.



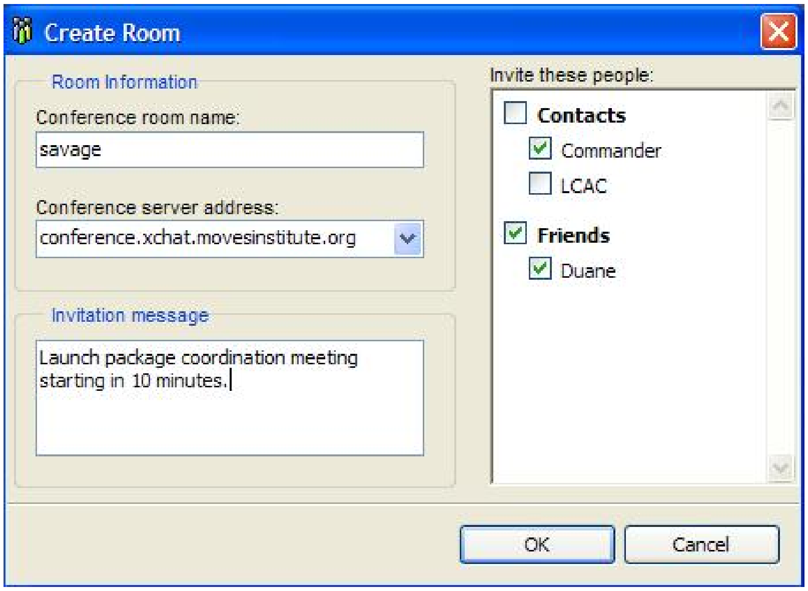
▽ Public Chatrooms
savage (0/30)
x3d (0/30)
xj3d (1/30)
xml (0/30)

▷ Public Chatrooms

▽ Public Chatrooms
!_Ekipa_! (0/30)
#beewolf (2/30)
#df (1)
#flashmob.spb.ru%irc.dalnet.ru (0/30)
acidrulez (0/30)
adonisys (0/20)
ArcaneWorld (0)

Figure 28. RhymBox screen shot listing available Conference rooms.

Next is the screen used to create a conference room and allow the user invite contacts when a room is created.



Conference room name: savage

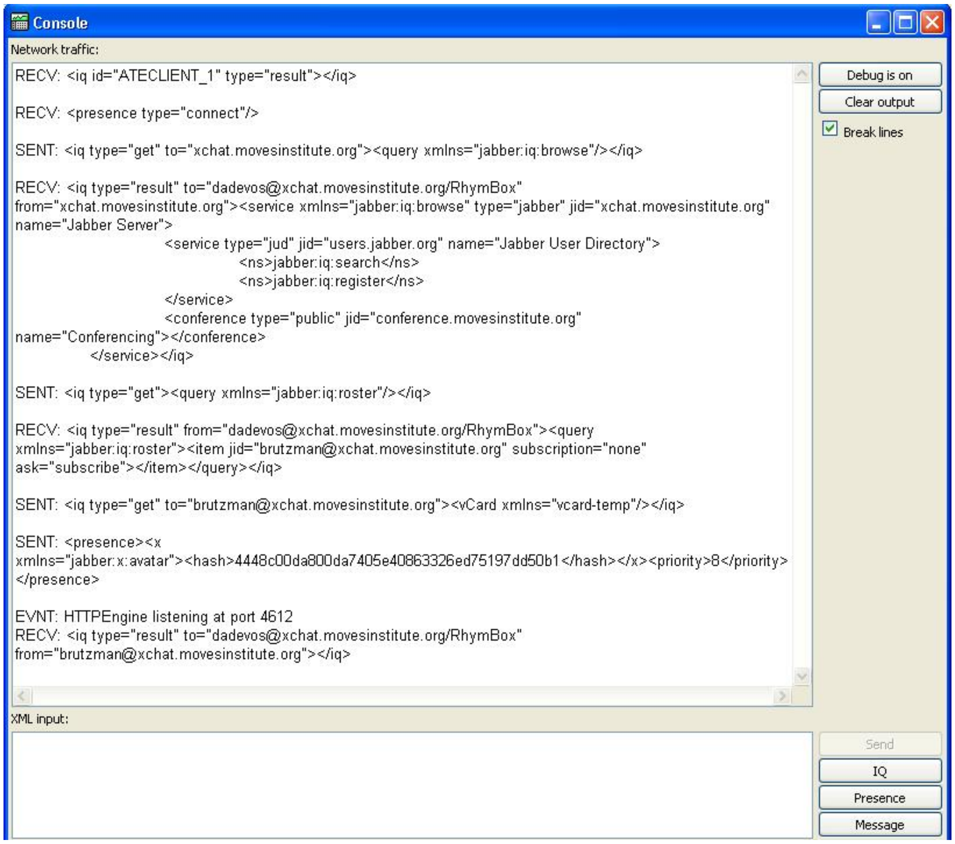
Conference server address: conference.xchat.movesinstitute.org

Invitation message: Launch package coordination meeting starting in 10 minutes.

□ Contacts
☑ Commander
□ LCAC
☑ Friends
☑ Duane

Figure 29. Rhymbox screen used to create a conference room.

The last screen shots show the debug feature of Rhymbox. This example shows the XML traffic when a user logs into the Jabber server. Note that `SENT:` and `RECV:` are interspersed by the client for display purposes only. Only XML tags are sent over the chat channel.



```
RECV: <iq id="ATECLIENT_1" type="result"></iq>

RECV: <presence type="connect"/>

SENT: <iq type="get" to="xchat.movesinstitute.org"><query xmlns="jabber:iq:browse"/></iq>

RECV: <iq type="result" to="dadevos@xchat.movesinstitute.org/RhymBox"
from="xchat.movesinstitute.org"><service xmlns="jabber:iq:browse" type="jabber" jid="xchat.movesinstitute.org"
name="Jabber Server">
                <service type="jud" jid="users.jabber.org" name="Jabber User Directory">
                        <ns>jabber:iq:search</ns>
                        <ns>jabber:iq:register</ns>
                </service>
                <conference type="public" jid="conference.movesinstitute.org"
name="Conferencing"></conference>
        </service></iq>

SENT: <iq type="get"><query xmlns="jabber:iq:roster"/></iq>

RECV: <iq type="result" from="dadevos@xchat.movesinstitute.org/RhymBox"><query
xmlns="jabber:iq:roster"><item jid="brutzman@xchat.movesinstitute.org" subscription="none"
ask="subscribe"></item></query></iq>

SENT: <iq type="get" to="brutzman@xchat.movesinstitute.org"><vCard xmlns="vcard-temp"/></iq>

SENT: <presence><x
xmlns="jabber:x:avatar"><hash>4448c00da800da7405e40863326ed75197dd50b1</hash></x><priority>8</priority>
</presence>

EVNT: HTTPEngine listening at port 4612
RECV: <iq type="result" to="dadevos@xchat.movesinstitute.org/RhymBox"
from="brutzman@xchat.movesinstitute.org"></iq>
```

Figure 30. Debugging a chat session with RhymBox reveals the underlying XML chat messages that are sent and received.

The features of RhymBox are show in Figure 31.

- Choice of chat or message style
- Typing notification
- File transfer ability
- Customizable avatars
- Customizable emoticons (hundreds included)
- Text conferencing (group chat)
- Message history logging
- Interoperability with MSN, Yahoo, AIM and ICQ
- Sound alerts
- Multiple-profile settings
- Auto-reconnection
- SOCKS proxy support
- Secure encryption (SSL)
- No adware or spyware
- No cost

Figure 31. List of advantageous RhymBox features.

This chat client is easy to download and install; setup is intuitive and all features work well. Of the three chat engines examined in this report, RhymBox is easiest to use.

7. Exodus

Exodus is a Jabber client for chat with a Jabber server and can be found at http://exodus.jabberstudio.org. Exodus has a GNU Public License (GPL) and is written in Delphi. The first screen in Exodus is the login screen. When creating a new account, Exodus asks user for additional information. This feature is optional to Jabber, and can be done at a later time.

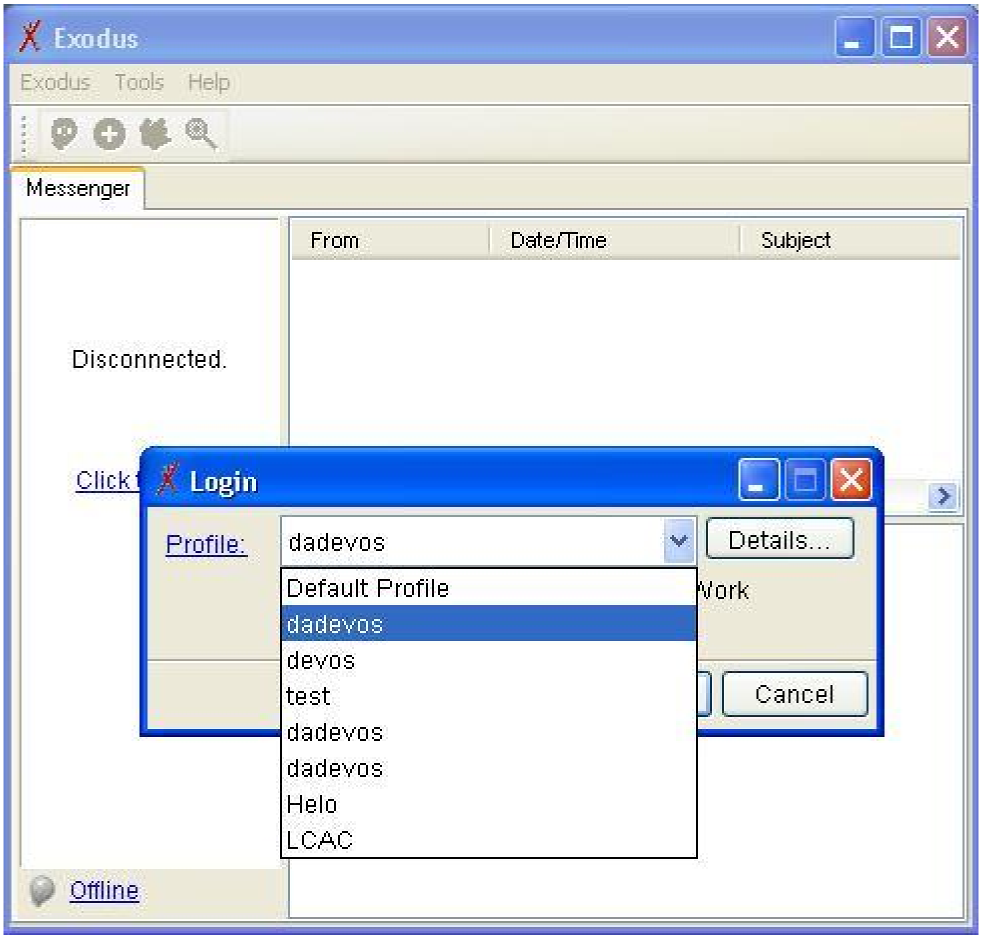
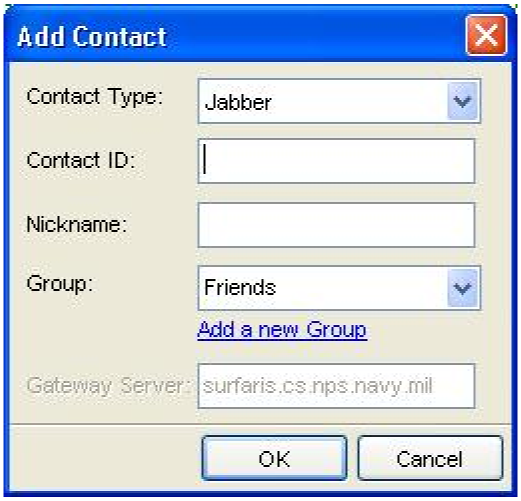


Figure 32. Multiple profiles let users share Exodus with others on the same computer or have accounts on multiple servers.

The next step is the Add a Contact screen, in which the user adds contacts by entering the contacts JID in the `Contact ID` box, along with nicknames. Exodus also provides group contacts.



Contact Type: Jabber

Contact ID:

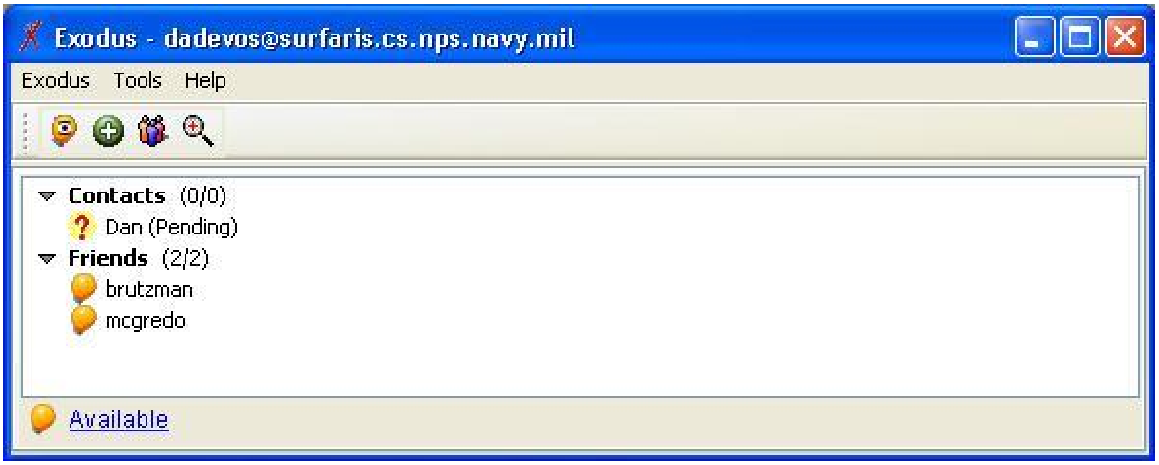
Nickname:

Group: Friends

Gateway Server: surfaris.cs.nps.navy.mil

Figure 33. Screenshot of the Add Contact feature in Exodus.

The next screen shows all user contacts and their on-line status. A yellow light bulb indicates that the contact is available and on-line. A question mark indicates the contact has not yet accepted the invitation to be a buddy.

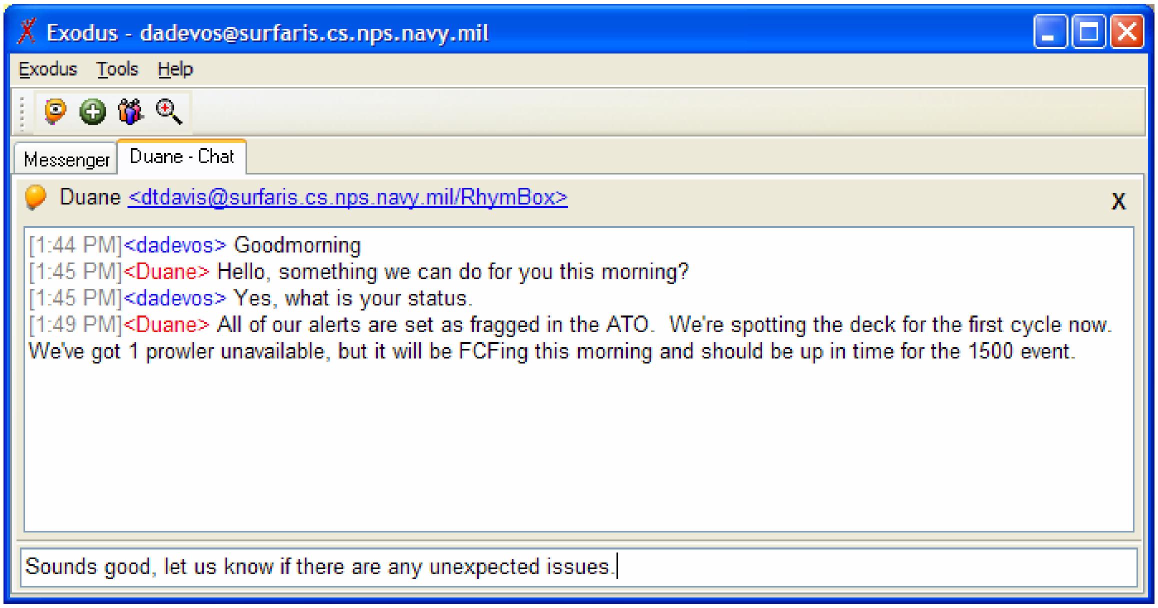


Contacts (0/0)
- ? Dan (Pending)

Friends (2/2)
- brutzman
- mcgredo

Figure 34. Screenshot showing contacts' online status in Exodus.

The next screen is the standard chat window for Exodus.

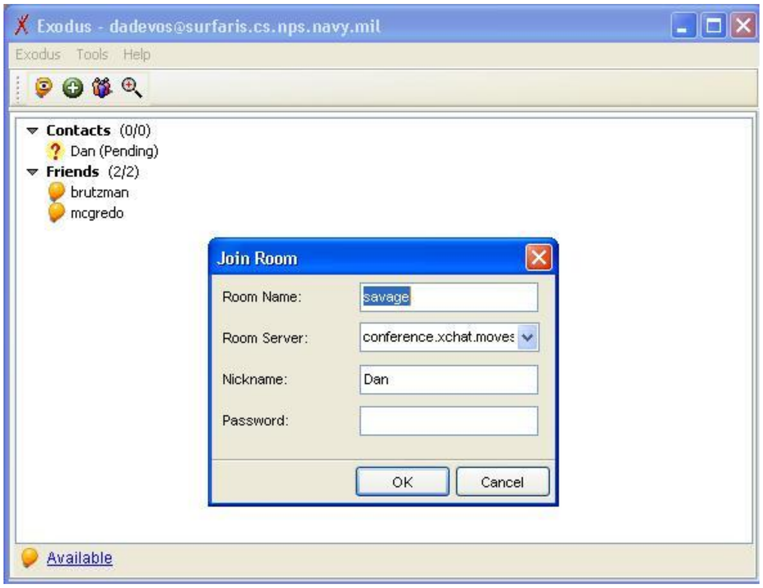

Exodus - dadevos@surfaris.cs.nps.navy.mil
Exodus Tools Help
Messenger
Duane - Chat
Duane <dtdavis@surfaris.cs.nps.navy.mil/RhymBox>
X

[1:44 PM]<dadevos> Goodmorning
[1:45 PM]<Duane> Hello, something we can do for you this morning?
[1:45 PM]<dadevos> Yes, what is your status.
[1:49 PM]<Duane> All of our alerts are set as fragged in the ATO. We're spotting the deck for the first cycle now. We've got 1 prowler unavailable, but it will be FCFing this morning and should be up in time for the 1500 event.

Sounds good, let us know if there are any unexpected issues.

Figure 35. Example of a P2P chat session in Exodus.

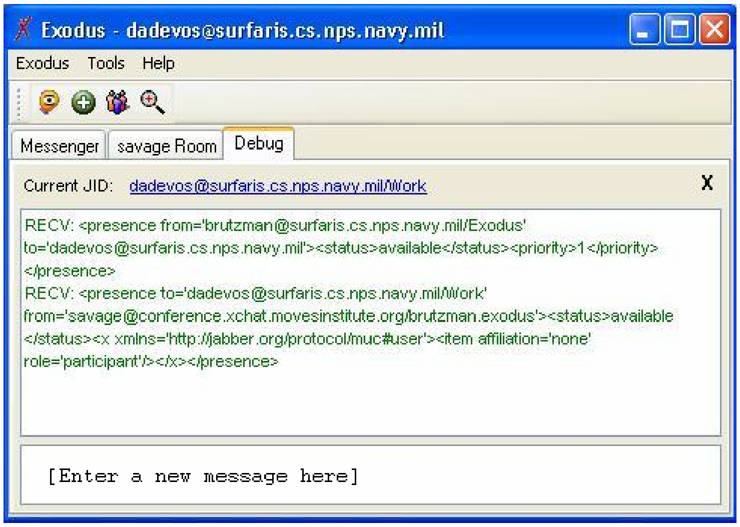
At this time, Exodus does not list conference rooms available. The next screen is used for joining a conference room, requiring the user to enter the room name, server, and a nickname for use in the room. The password block is for the room password, and should not be entered unless the room was password-protected when created.



Room Name: savage
Room Server: conference.xchat.moves
Nickname: Dan
Password:

Figure 36. Screenshot of the logon procedure for joining a chat room in Exodus.

The next screen shots show the debug feature of Exodus. This example shows the XML traffic when a user's presence is updated.

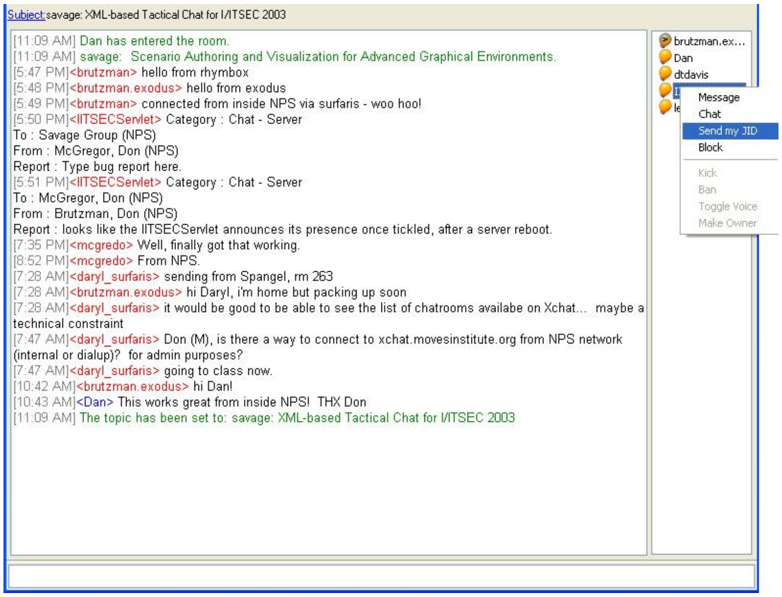

Exodus - dadevos@surfaris.cs.nps.navy.mil
Exodus
Tools
Help
Messenger
savage Room
Debug
X
[Enter a new message here]

Current JID: dadevos@surfaris.cs.nps.navy.mil/Work

```
RECV: <presence from='brutzman@surfaris.cs.nps.navy.mil/Exodus'
to='dadevos@surfaris.cs.nps.navy.mil'><status>available</status><priority>1</priority>
</presence>
RECV: <presence to='dadevos@surfaris.cs.nps.navy.mil/Work'
from='savage@conference.xchat.movesinstitute.org/brutzman.exodus'><status>available
</status><x xmlns='http://jabber.org/protocol/muc#user'><item affiliation='none'
role='participant'/></x></presence>
```

Figure 37. Debugging a chat session with Exodus.

Two nice features of Exodus are the tab feature for displaying multiple windows and the ability to add contacts from the conference room screen by right-clicking their name and sending them a JID. Below are screenshots depicting these features.

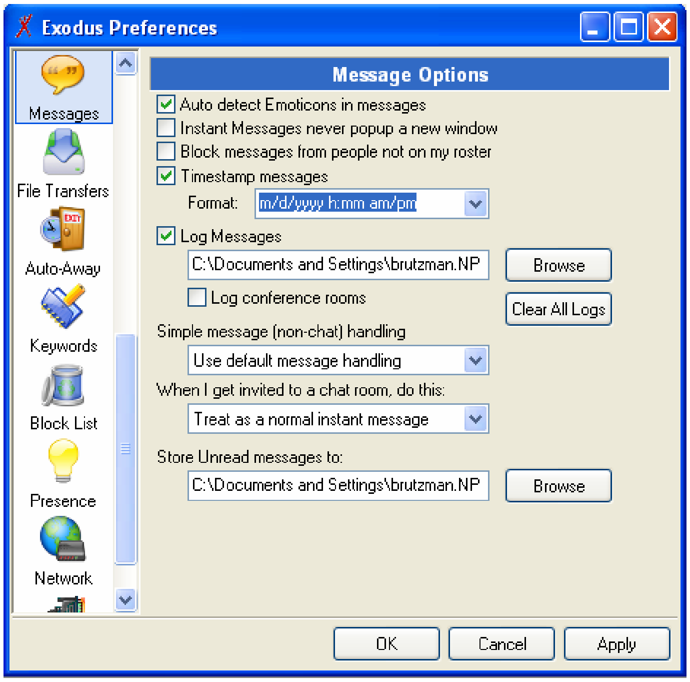


Subject: savage: XML-based Tactical Chat for I/ITSEC 2003

```
[11:09 AM] Dan has entered the room.
[11:09 AM] savage:  Scenario Authoring and Visualization for Advanced Graphical Environments.
[5:47 PM]<brutzman> hello from rhymbox
[5:48 PM]<brutzman.exodus> hello from exodus
[5:49 PM]<brutzman> connected from inside NPS via surfaris - woo hoo!
[5:50 PM]<IITSECServlet> Category : Chat - Server
To : Savage Group (NPS)
From : McGregor, Don (NPS)
Report : Type bug report here.
[5:51 PM]<IITSECServlet> Category : Chat - Server
To : McGregor, Don (NPS)
From : Brutzman, Don (NPS)
Report : looks like the IITSECServlet announces its presence once tickled, after a server reboot.
[7:35 PM]<mcgredo> Well, finally got that working.
[8:52 PM]<mcgredo> From NPS.
[7:28 AM]<daryl_surfaris> sending from Spangel, rm 263
[7:28 AM]<brutzman.exodus> hi Daryl, i'm home but packing up soon
[7:28 AM]<daryl_surfaris> it would be good to be able to see the list of chatrooms availabe on Xchat...  maybe a
technical constraint
[7:47 AM]<daryl_surfaris> Don (M), is there a way to connect to xchat.movesinstitute.org from NPS network
(internal or dialup)?  for admin purposes?
[7:47 AM]<daryl_surfaris> going to class now.
[10:42 AM]<brutzman.exodus> hi Dan!
[10:43 AM]<Dan> This works great from inside NPS!  THX Don
[11:09 AM] The topic has been set to: savage: XML-based Tactical Chat for I/ITSEC 2003
```

Figure 38. Add a contact while in a conference room by right-clicking his name and sending him a JID.

The feature in Exodus that sets it apart from Rhymbox is the ability to turn on timestamps. At the Tools/Preferences/Messages/Message Options tab, check the Timestamp Messages box. To set the timestamp to display a date such as 12/18/03 8:26 am, type the following undocumented format into the Format field [`m/d/y h:mm am/pm`], as seen in figure 39.



Message Options

☑ Auto detect Emoticons in messages

□ Instant Messages never popup a new window

□ Block messages from people not on my roster

☑ Timestamp messages

Format: m/d/yyyy h:mm am/pm

☑ Log Messages

C:\Documents and Settings\brutzman.NP

□ Log conference rooms

Simple message (non-chat) handling

Use default message handling

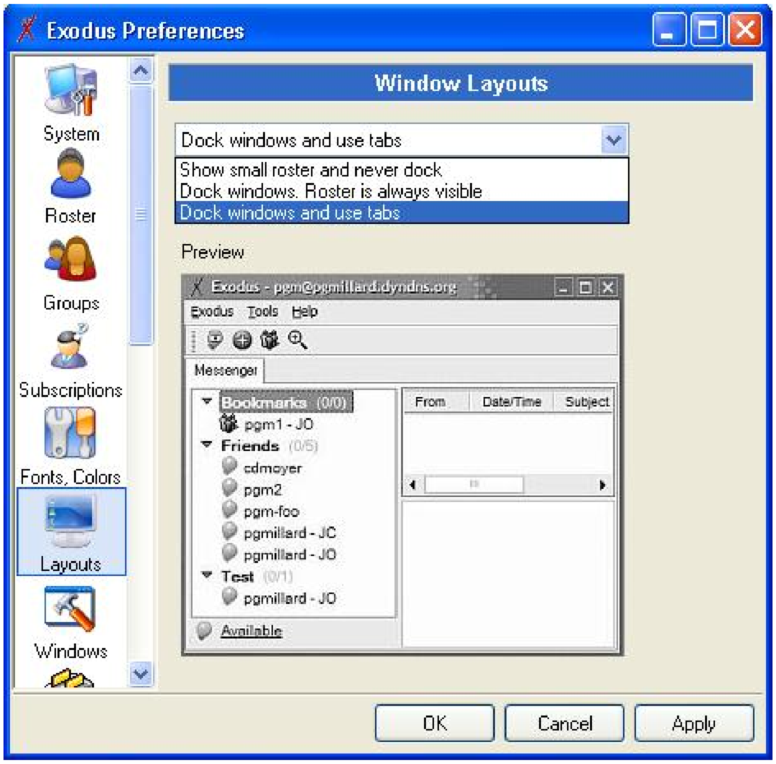
When I get invited to a chat room, do this:

Treat as a normal instant message

Store Unread messages to:

C:\Documents and Settings\brutzman.NP

Figure 39. Turn on the timestamping feature in Exodus to date chat messages.



Window Layouts

Dock windows and use tabs

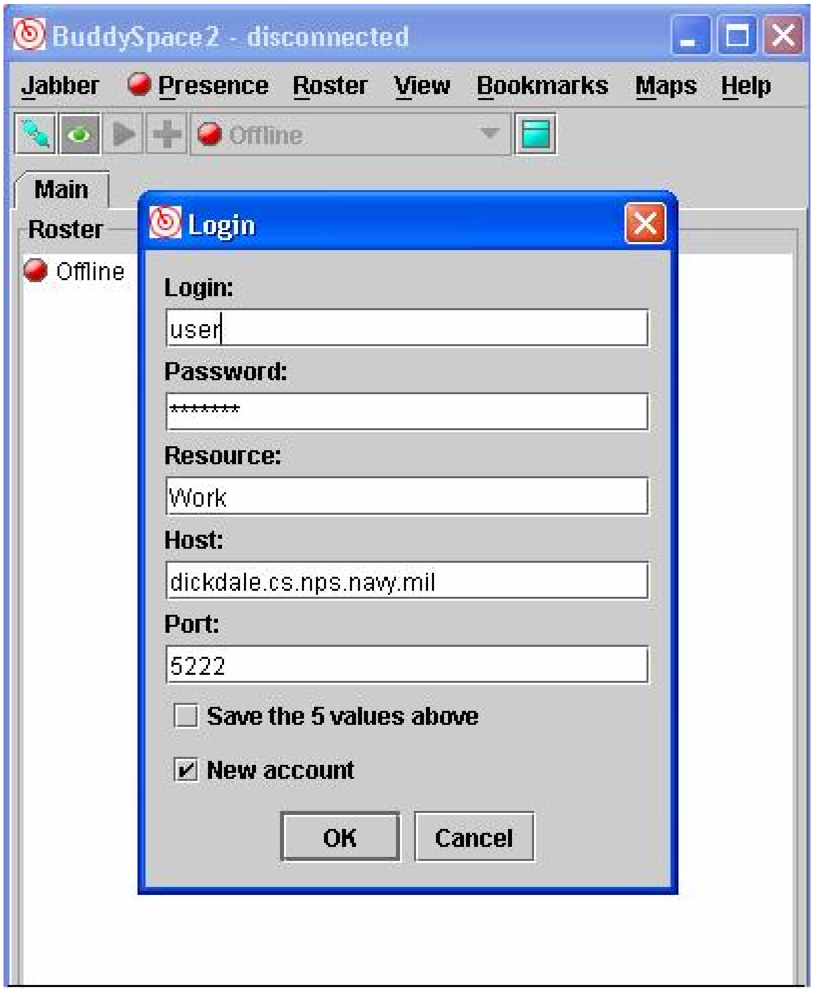
Figure 40. Selecting the Dock Windows and Use Tabs option in Exodus layers windows, which organizes the chat window and allows easy switching between sessions.

To use SSL connections with Exodus, the SSL DLLs need to be downloaded. This zip package contains two DLLs, which must be placed in the directory where Exodus is installed. To minimize download size, these DLLs are not included by default. The DLLs can be found at http://exodus.jabberstudio.org/indy_openssl096g.zip (or simply via http://exodus.jabberstudio.org).

Other helpful features are the *echo* command, which repeats what is typed, and automatic logging, which saves chat history in HTML with time stamps.

8. BuddySpace

BuddySpace is a Jabber client for chatting with a Jabber server, found at http://buddyspace.sourceforge.net. BuddySpace has a Jabber open-source license (JOSL) and is written in Java, which is of interest for possible software customization. The BuddySpace initial login screen requires the user to retype his information each time he changes accounts, which is not a good feature.



Login:
user

Password:
****

Resource:
Work

Host:
dickdale.cs.nps.navy.mil

Port:
5222

□ Save the 5 values above

☑ New account

Figure 41. The BuddySpace login screen requires the user to retype all information when switching accounts.

When one tries to open a conference room with BuddySpace, the error "Error 501: Not Implemented!" appears. According to David Sutton (screen name “peregrine”), the creator of MU-conference protocol, BuddySpace is currently using the “conferencing” protocol to connect to conference rooms. This was never endorsed as a protocol, although the conference v0.4 jabber component does accept it. It does not have the same functionality as “groupchat,” however, as groupchat doesn't use the jabber:iq:conference calls to check and request rooms. If it were supporting groupchat, it would connect quite happily to MU-conference, as it was designed to handle both MUC and groupchat. [Sutton 03]

BuddySpace does not support SSL at present.

9. Observations from Using Clients

All clients feature a debug window that allows the user to view the XML network traffic. Both Exodus and RhymBox allow the user to join conference rooms. BuddySpace offers to let the user join a room, but says “not implemented” when the attempt is made, and fails to admit him. BuddySpace is not ready for tactical-chat use.

C. JABBER SERVER SET-UP

The experimental Jabber servers at NPS are on `surfaris.cs.nps.navy.mil` and `dickdale.cs.nps.navy.mil`. They are behind the NPS firewall and can't be accessed from outside. Outside the firewall, two servers are available for testing at `xchat.movesinstitute.org` on port `5226` or `jabber.metnet.navy.mil` on port `8080`.

This report details how servers were set up on Surfaris and xchat. Services covered include conference rooms, firewalls, logging and server to server (S2S). The following diagram shows the initial server setup.

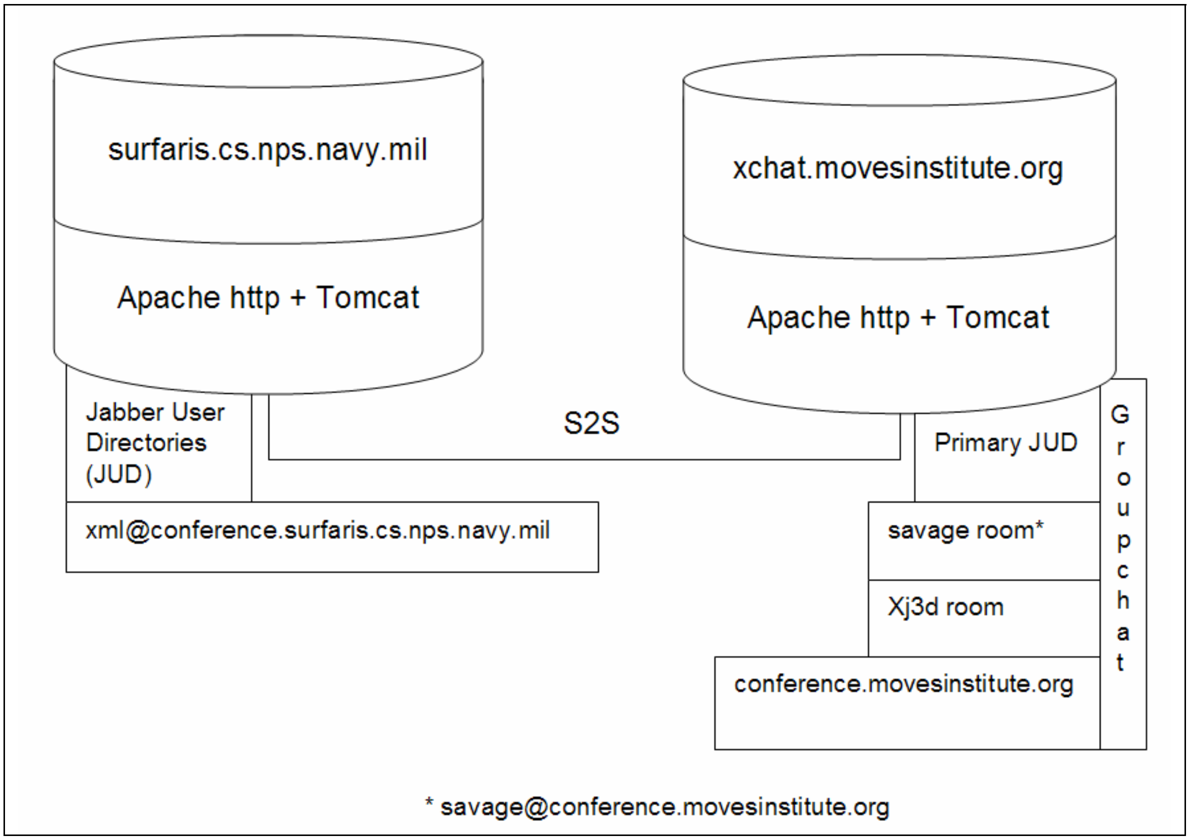

surfaris.cs.nps.navy.mil
Apache http + Tomcat
xchat.movesinstitute.org
Apache http + Tomcat
Jabber User Directories (JUD)
S2S
Primary JUD
Groupchat
xml@conference.surfaris.cs.nps.navy.mil
savage room*
Xj3d room
conference.movesinstitute.org
* savage@conference.movesinstitute.org

Figure 42. Setup summary of chat servers and conference rooms on `surfaris` and `xchat`.

1. Setting Up a Server on `Surfaris`

NPS first attempted to install the Jabber server OpenIM on Windows, which failed. The next attempt was to install jabberd 1.4.2 on Linux, which seems to be the most widely used Jabber server. The configuration followed the usual UNIX process of typing "configure" in the distribution directory. This examines the machine and builds a makefile. Once this is done, the user types "make" to compile the program, and "make install" to install it.

Jabberd uses one of several possible databases to manage data. The options include MySQL and Berkeley DB. This report uses Berkeley DB. The version of Berkeley database required was quite recent, 4.2.52 or later. The rpm file was downloaded from sleepycat software, which maintains an open-source version of the Berkeley DB, and is installed it in the usual way (`rpm -i anRPMFile.rpm`).

Server setup requires making a directory for configuration, setting a host name, and encryption software if encryption is desired. Configuration is done in part via the files `/etc/sysconfit/jabber`. The SSL feature, which allows encrypted communications, was turned off, but is installed on dickdale. To get this feature, openSSL needs to be install. Other configuration files include `/usr/local/jabberd-1.4.2/jabber.xml`.

The directory `/usr/local/jabberd-1.4.2` has translators for various chat programs, including AOL's and ICQ's. Other plug-ins can be placed in this directory including one for conference rooms.

The server process is started via `/etc/init.d/jabberd`. This process is called automatically at startup, or can be manually started or stopped by directly executing the script.

Jabberd 2.0 was not released in stable form at time of the initial installation and test on `surfaris` (January 2004).

2. Setting Up a Server on MovesInstutute.org

Clients and servers from outside the firewall cannot connect with `safaris` at this time. A second Jabber server was installed on movesinstitute.org. Attempts to configure both servers in like fashion didn't work.

A few notes from the setup of xchat.movesinstitute.org:

1. DNS

The xchat name needed to be paired with an IP number. The DNS provider for movesinstitute.org is zoneedit.com, who provide a nice Web-based interface for manipulating DNS tables: DNS simply matches a name with an IP number. The location of the IP number is irrelevant. The following entry was made to the list of hosts in the MovesInstitute.org domain:

Host Name	IP
confernce.xchat.movesinstute.org	12.213.189.186
xchat.movesinstitute.org	12.213.189.186

Figure 43. IP addresses needed to install Jabber server on MovesInstitute.org.

The 12.213.189.186 IP is dynamically assigned by comcast.net to a home computer via DHCP. DHCP-assigned IPs can be fairly stable over long periods of time in the right conditions. Comcast has their DHCP server configured to hand out leases for four days' duration, and because of the way the DHCP protocol works, this IP is constantly renewed.

2. Firewall

The `xchat` MovesInstitute.org machine contains a firewall. Since it is exposed to the Internet at large, it constantly fields probes from people or worms looking for a way in. This usually manifests itself as probes for IIS weaknesses, but other port scans go on as well. For example, putting up a Web server shows these log entries:

```
     12.229.49.123 - - [02/Sep/2003:14:01:01 -0700] "GET
/scripts/..%c1%1c../winnt/system32/cmd.exe?/c+dir HTTP/1.0"
404 304
```

These are attempts to use unicode exploits to compromise a Windows box running IIS. Buffer-overrun exploits are also continually attempted.

Modern Linux boxes use iptables firewalls. In this case, an iptables script was configured at /etc/rc.d/rd.firewall-2.4 that shut off all access to ports on the machine except for those explicitly allowed. The following services were allowed:

```
     echo -e "   - Allowing EXTERNAL access to the WWW
server"
     $IPTABLES -A INPUT -i $EXTIF -m state --state
NEW,ESTABLISHED,RELATED \ -p tcp -s $UNIVERSE -d $EXTIP --
dport 80 -j ACCEPT

     # Let SSH come in
     echo -e "   - Allowing EXTERNAL access to SSH"
     $IPTABLES -A INPUT -i $EXTIF -m state --state
NEW,ESTABLISHED,RELATED \ -p tcp -s $UNIVERSE -d $EXTIP --
dport 22 -j ACCEPT

     # Let jabber come in
     echo -e "   - Allowing EXTERNAL access to jabber"
     $IPTABLES -A INPUT -i $EXTIF -m state --state
NEW,ESTABLISHED,RELATED \ -p tcp -s $UNIVERSE -d $EXTIP --
dport 5222 -j ACCEPT

     # Let jabber ssl come in
     echo -e "   - Allowing EXTERNAL access to jabber via
SSL"
```

```
     $IPTABLES -A INPUT -i $EXTIF -m state --state
NEW,ESTABLISHED,RELATED \ -p tcp -s $UNIVERSE -d $EXTIP --
dport 5223 -j ACCEPT
```

The firewall script was run from the init.d directory script firewall-2.4, which was turned on via the chkconfig command.

3. Jabberd

Jabber itself was installed at `/usr/local/jabber`. The main configuration file is jabber.xml. The jabberd daemon is started via the jabberd script in `/etc/init.d`, or via the chkconfig command. The compilation of jabberd is labor intense. In the jabber directory, type:

[jabber]# ./configure --enable-openssl --with-openssl=/usr/local/ssl/usr/local/ssl

This directory contains Openssl 0.9.6. Jabber is picky about the version of SSL used, returning compilation errors if a more modern version, such as that installed on Linux by default, is attempted.

Jabberd runs from the compilation directory. Jabberd is compiled locally in `/usr/local/jabberd-1.4.2` and all the executables are in that directory.

3. Changes to jabber.xml

The jabber.org page lists different example jabber.xml pages and is a good resource for comparison. http://www.jabber.org/admin/config-examples.php. This report uses the jabberd daemon 1.4.2 and not the experimental jabberd version 2. The following are suggested changes to the `jabber.xml` configuration file.

```
<!-- The server vCard -->
<vCard>
<FN>Jabber Server</FN>
   <DESC>A Jabber Server!</DESC>
   <URL>http://movesinstitute.org/</URL>
</vCard>

to

<!-- The server vCard -->
<vCard>
   <FN>xchat.MovesInstitute.org Jabber Server</FN>
   <DESC>xchat MOVES Institute Jabber Server!</DESC>
   <URL>http://xchat. MovesInstitute.org</URL>
</vCard>
```

Figure 44. Changes to `jabber.xml` when implementing `jabberd` 1.4.2.

The following XML in jabber.xml is the resource that checks for updated versions of the Jabber server software. Note that no functionality is lost if this is commented out. Removing the <update/> tag is a good strategy if the Jabber server is behind a firewall. To use this feature, change "localhost" to the hostname or IP address of the server, making sure that it is the same as the entry for <host/> above.

```
<update>
   <jabberd:cmdline flag="h">localhost</jabberd:cmdline>
</update>

to

<update>
   <jabberd:cmdline flag="h">xchat.movesinstitute.org</jabberd:cmdline>
</update>
```

Figure 45. Changes to jabber.xml needed to implement version updates in jabberd.

The following XML in jabber.xml is the default server record-logging component, which logs general statistical and tracking data. This code stores the log file in XML format.

```
<log id="rlogger">
   <host/>
   <logtype>record</logtype>
   <format>%d %h %s</format>
   <file>record.log</file>
</log>

to

<log id="rlogger">
   <host/>
   <logtype>record</logtype>
   <format>&lt;message@time=%d @host=%h&gt;%s&lt;/message&gt;</format>
   <file>record.log.xml</file>
</log>
```

Figure 46. Changes to jabber.xml needed to implement XML formatted logging in jabberd.

An IRC client is available at http://www.mirc.org. When the IRC component is ready, the following section shown in figure 47 must be uncommented. Current efforts to get IRC working with a Jabber server have been unsuccessful at this point.

```
<service id="irc">
   <host>irc.localhost</host>
</service>
```

Figure 47. Changes to `jabber.xml` needed to implement IRC in `jabberd`.

The procedure for connecting xchat to surfaris is covered by "Splitting Up Jabber Server Processes" in the O'Reilly book, pp. 113-118. [Monson 01] This was completed by making the required setting changes in `jabber.xml` on both servers. The connection uses the default port for server-to-server connections is 5269.

Future work involves installing Jabberd 2.x and component services. The 2.x beta was in not stable, so the fallback was the 1.4.2 release on `surfaris` and 1.4.3 release on `xchat`. Open-source component services for Jabber including SMTP, SMS and gateways for AIM, MSN, and Yahho are found at http://www.jabber.org/software/components.php.

D. SETTING UP WEB HTTP SERVER AND JAVA SERVLETS

1. Solutions for HTTP Post

There are three ways to perform an HTTP post from a browser to a Web server/Jabber server: an HTML form post, an XML-HTTP object, and an applet. In the HTTP form-post method, the browser wraps the XML string into the value portion of an `element=”value”` pair and posts it to the Web server. At the server end, the XML data is extracted from the `element=”value”` pair. Extraction-code languages can be Perl, active server pages (ASP), Java server pages (JSP) or Java servlets.

The XMLHTTP object method uses direct posting of XML data to a Jabber server using HTTP post. We rejected this method because it involves an ActiveX object, and is thus a Windows-only solution. One possibility using applets is to incorporate design time controls (DTCs) from Visual Interdev 6.0. [Perkins 03]

Using applets is possible, but the use of Java and Javascript on the client side was undesirable. HTTP form post is therefore the method chosen.

2. Converting Post to XML

The next point is what technology is needed to convert the post to XML: Javascript client, server-side CGI or servlet. This report uses servlet technology because it is supported on many different servers and makes the smallest program for the client. The steps that the servlet must handle are:

• Client posts to server.
• Java servlet receives from client.
• Java servlet process incoming XHTML Form data (XML Tactical Chat messages).
• Servlet pulls XML from post.
• Servlet sends posts as a Jabber message to the Jabber server, with proper XML wrapper.
• The Jabber server in turn sends message to JABBER clients (Exodus, Rhymbox or BuddySpace).

Figure 48. Functions performed by the Post to XML servlet

3. Jabber Parameter Names

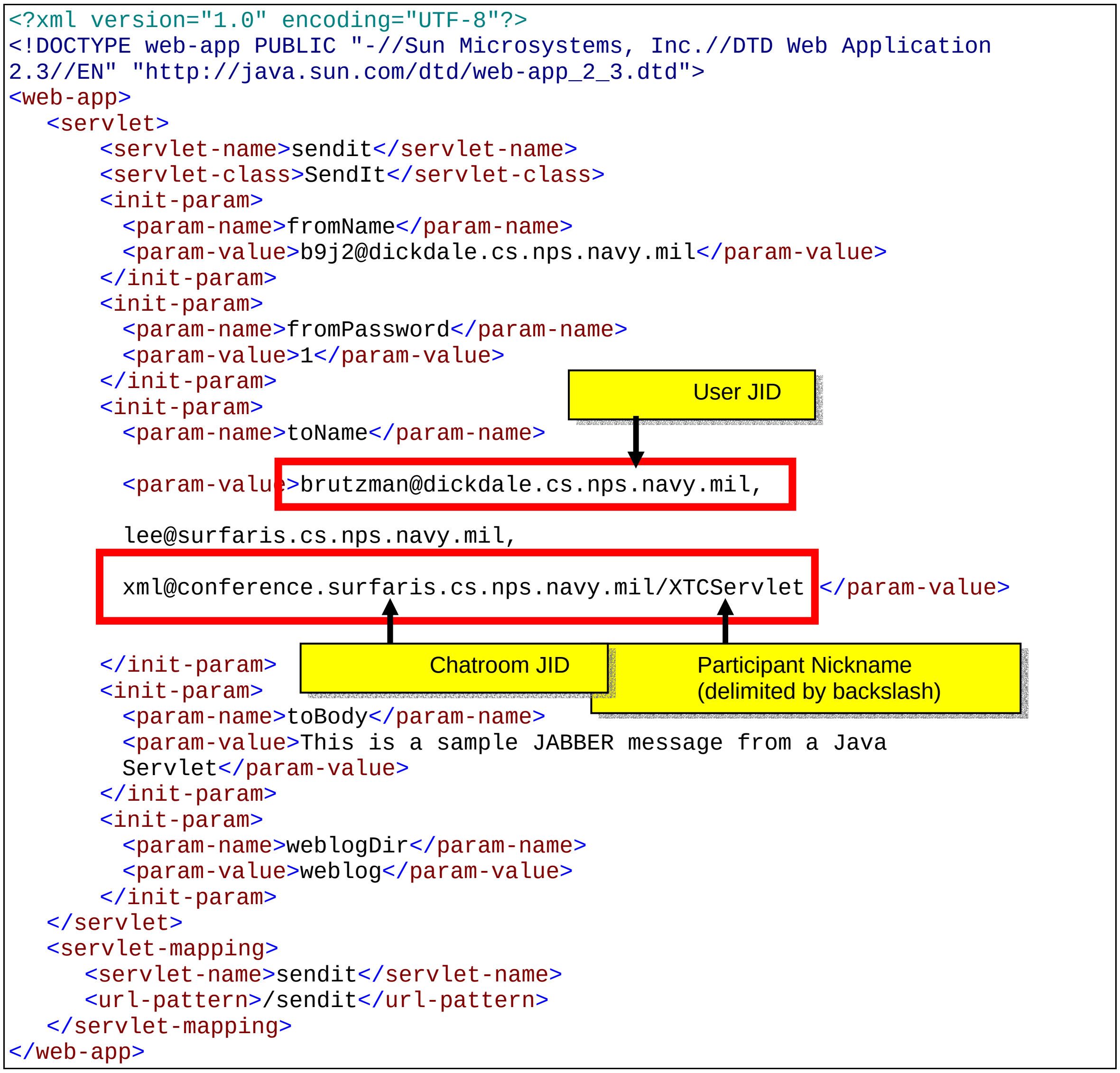
Jabber uses specific parameters that must be set in the `web.xml` file. Figure 49 lists those parameters and gives a brief description.

Param-name	Param-name	Description
fromName	B9j2@dickdale.cs.nps.navy.mil	Sender account name. One entry only.
fromPassword	<password>	Password of sender. One entry only.
toName	lee@dickdale.cs.nps.navy.mil, brutzman@dickdale.cs.nps.navy.mil	Who to send to. Multiple entries delimited by commas. Spaces are automatically trimmed.
toBody	This is a sample JABBER message from a Java Servlet	Default message body if none is provided.
weblogDir	weblog	Directory to save generated XML messages.

Figure 49. Required parameters for Jabber to be set in the `web.xml` file.

4. Web Server Configuration – `web.xml`

The `web.xml` file tells the servlet what parameters to expect from the XHTML form. Below is the `web.xml` file with the toName parameter highlighted to show the format for a JID.

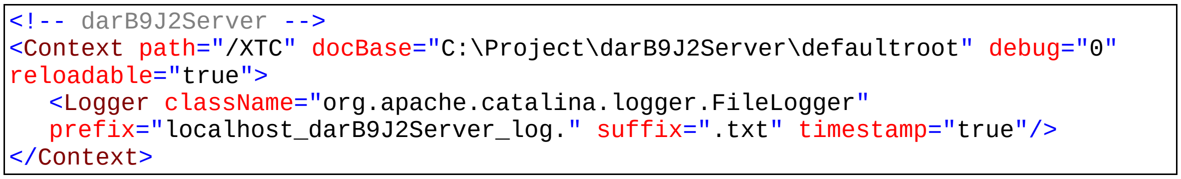


```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application
2.3//EN" "http://java.sun.com/dtd/web-app_2_3.dtd">
<web-app>
   <servlet>
       <servlet-name>sendit</servlet-name>
       <servlet-class>SendIt</servlet-class>
       <init-param>
         <param-name>fromName</param-name>
         <param-value>b9j2@dickdale.cs.nps.navy.mil</param-value>
       </init-param>
       <init-param>
         <param-name>fromPassword</param-name>
         <param-value>1</param-value>
       </init-param>
       <init-param>
         <param-name>toName</param-name>

         <param-value>brutzman@dickdale.cs.nps.navy.mil,

         lee@surfaris.cs.nps.navy.mil,

         xml@conference.surfaris.cs.nps.navy.mil/XTCServlet </param-value>

       </init-param>
       <init-param>
         <param-name>toBody</param-name>
         <param-value>This is a sample JABBER message from a Java
         Servlet</param-value>
       </init-param>
       <init-param>
         <param-name>weblogDir</param-name>
         <param-value>weblog</param-value>
       </init-param>
   </servlet>
   <servlet-mapping>
      <servlet-name>sendit</servlet-name>
      <url-pattern>/sendit</url-pattern>
   </servlet-mapping>
</web-app>
```

Figure 50. The `web.xml` configuration file that configures the XTC servlet.

5. Web Server Configuration – server.xml

This report uses Tomcat as the Web server. To enable logging on, the following update needs to be made to the server.xml file.



```
<!-- darB9J2Server -->
<Context path="/XTC" docBase="C:\Project\darB9J2Server\defaultroot" debug="0"
reloadable="true">
   <Logger className="org.apache.catalina.logger.FileLogger"
   prefix="localhost_darB9J2Server_log." suffix=".txt" timestamp="true"/>
</Context>
```

Figure 51. Changes to the server.xml file that enable log-on the Tomcat server.

6. Servlet Configuration: Peer to Peer

The following Java servlet is needed to send a Jabber message through the Web server to a Jabber server.

```
//-----------------------------------------------------------------------------
/**
 *
 * @param accountName login user name (e.g. myaccount)
 * @param serverName login server (e.g. dickdale.cs.nps.navy.mil)
 * @param passWord login user password
 * @param toAccount which account to send to (e.g. anotheraccount)
 * @param toServer which server (e.g. dickdale.cs.nps.navy.mil)
 * @param messageBody body
 * @return success=true, failure=false
 */
public String sendMsgP2P(String accountName, String serverName, String
       passWord, String toAccount, String toServer, String messageBody) {
       // Setup a connection bean.
        ……
       //-----------------------------------------------------------------------
       // Send the message
       MessageBuilder mb=new MessageBuilder();
       Message msg;
       // Who and where to send the message to.
       mb.setToAddress(new JID(toAccount, toServer, null));
       mb.setSubject( MSG_SUBJECT_P2P +" "+ SystemUtil.getDateTime14() );
       mb.setBody(messageBody);
       try {
              msg=(Message)mb.build();
       }
       catch (InstantiationException e) {
              // If the build failed...
       }
       //Send the message!
       cb.send(msg);
       return “GOOD”;

} /* sendMsgP2P */
```

Figure 52. Servlet source code that enables a message to be sent P2P.

7. Servlet Configuration – Chat room

The following Java servlet sends a Jabber message through the Web server to a chat room on a Jabber server.

```
//---------------------------------------------------------------------------
/**
 * send JABBER message to chatroom
 *
 * @param accountName login user name (e.g. myaccount)
 * @param serverName login server (e.g. dickdale.cs.nps.navy.mil)
 * @param passWord login user password
 * @param nickName nick name to use in chatroom
 * @param toAccount which account to send to (e.g. anotheraccount)
 * @param toServer which server (e.g. dickdale.cs.nps.navy.mil)
 * @param messageBody body
 * @return success=true, failure=false
 */
public String sendMsgChatRoom(String accountName, String serverName, String
       passWord, String nickName, String toAccount, String toServer, String
       messageBody) {
       // Setup a connection bean.
             ……
       //-----------------------------------------------------------------------
       // Send Presence
       sendPresence(cb, new JID(toAccount, toServer, nickName), null,
       "available", "available");
       //Add the room address to the list
       PresenceBean pbean = new PresenceBean();
       pbean.setConnBean(cb);
       //-----------------------------------------------------------------------
       // Send to Group Chat
       GroupChatBean gcb = new GroupChatBean(cb, new JID(toAccount, toServer,
       nickName), pbean);
       if ( gcb.sendMessage(null, messageBody) ) {
             gcb.shutdown();
             return “SUCCESS”;
       }
       else {
             gcb.shutdown();
             return “FAILED”;
       }
} /* sendMsgChatRoom */
```

Figure 53. Servlet source code that enables a message to be sent to a chat room.

E. SUMMARY

This chapter defines the capabilities of the clients, servers, and XHTML needed to demonstrate a successful XML-based. Jabber tactical-chat system. It describes how to download, configure and install an off-the-shelf Jabber client and shows two installations of a jabberd server. Finally, this chapter shows how to configure a Web server to handle XHTML-to-Jabber servlets.

VI. EXAMPLE XTC APPLICATION USING JABBER PROTOCOLS

A. INTRODUCTION

This chapter describes the development approach used to design aN XML-based, Jabber-enabled, tactical-chat system. The first section discusses the fundamental use of XML, XHTML, and open-source software in this report. Next is a list of initial target capabilities for XTC, followed by an explanation of system design. Details about the XHTML forms and XML behind XTC are then given, followed by an outlined tactical scenario and suggestion regarding how XTC can be used. The chapter then examines policy issues and related administrative developments with XTC and Jabber.

B. FUNDAMENTAL TECHNOLOGIES AND DEVELOPMENT METHODOLOGY

This section describes the specific technologies and development methodology used to create XTC.

1. XML Basis

An XML-based tactical chat system that implements Jabber provides significant improvements over the proprietary incompatibilities of currently fielded systems. Because XML standardizes data, it can be extended to replace and incorporate all electronic correspondence, including military messaging, e-mail, and VTC into one customizable environment. Additionally, the structure imposed by exclusive use of schema-governed message formats provides for capability extension while maintaining current functionality.

2. Uses XHTML to Enable Thin Client Web Page Input

XHTML forms provide a technology to predefine the content of chat messages and label them for meaningful storage and searching. XHTML offers a thin client because it can be accessed via the Web, passing an XML document to a Web server that forwards it to a Jabber server. In this case, the Web server is an alternative to creating a thick client because it passes the XML itself. On the receiving side, the simplest solution is to use an open-source Jabber client to read the XML. Thanks to open-source Jabber protocols and a Web-based client, the system is easier to maintain and update than current proprietary systems. Future work might be to create a custom client that provides such filtering and display options.

3. Uses open-source software

The three approaches to developing and implementing chat clients are a proprietary chat server/client system, an open-source solution, or a customization. Proprietary solutions are too costly and inflexible for a technology that is changing as fast as chat. Custom solutions take long to develop and rely on the developer to ensure security. This report shows the benefits of using open-source software to design a solution that is both flexible and extensible.

C. INITIAL TARGET CAPABILITIES

This section describes the initial capabilities that XTC supports: Jabber client, Jabber server, and XML Forms. The Jabber client capabilities are the features of the Jabber client that are needed to support XTC, listed in Figure 54.

- Log into Jabber server
- Sending a message from a client to a client - P2P (peer to peer)
- Sending an XML chat message from a client to the server - A2P (application to peer)
- View XML code
- Join an existing conference room

Figure 54. Jabber client target capabilities.

The Jabber server capabilities are the features of the Jabber server that are needed to support XTC are listed in Figure 55.

- Support conference rooms
- Log messages on the server
- Send a message from one server to another server - A2A (application to application)
- Be able to run on diverse operating systems

Figure 55. Jabber server target capabilities

The XML-forms capabilities are the features of the Web server and XHTML forms that are needed to support XTC are listed in Figure 56.

- Display an XHTML form.
- Submit an XHTML form to a Web server.
- Forward an XML message from the Web server to the Jabber server.
- Use an off the shelf Jabber client to log into the server and view the message.
- Login/logout to Jabber server on each message.
- Add time stamps – from Javascript on client side, obtain "Jabber time" through call to server. Use Javascript client time and format.
- Allow Jabber-wrapped XML tactical message to be received by Jabber server; XML message appears to participants in group session.

Figure 56. XML forms target capabilities.

The capabilities of these three categories can also be displayed working together in a seven-step dataflow diagram. The first two steps show the Web server displaying the XForm to the client's Web browser. Next, the form posts the data to the Web server. In steps 4 and 5, the Web server extracts the XML data and sends it to the Jabber server. Steps 6 and 7 show the Jabber server sending the data to the Jabber clients.

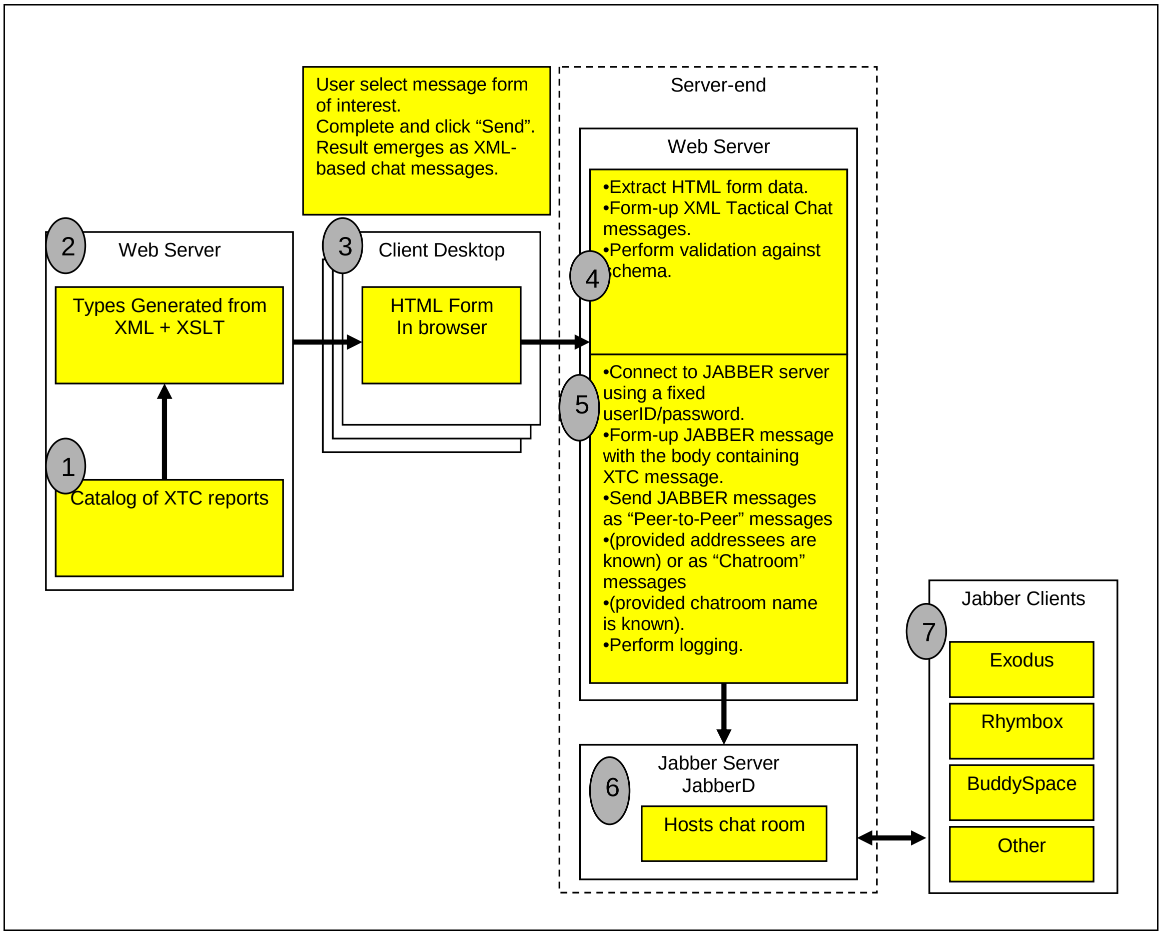

User select message form of interest.
Complete and click "Send".
Result emerges as XML-based chat messages.
Server-end
Web Server
•Extract HTML form data.
•Form-up XML Tactical Chat messages.
•Perform validation against schema.
2
Web Server
Types Generated from XML + XSLT
3
Client Desktop
HTML Form In browser
4
5
•Connect to JABBER server using a fixed userID/password.
•Form-up JABBER message with the body containing XTC message.
•Send JABBER messages as "Peer-to-Peer" messages
•(provided addressees are known) or as "Chatroom" messages
•(provided chatroom name is known).
•Perform logging.
1
Catalog of XTC reports
Jabber Clients
7
Exodus
Rhymbox
BuddySpace
Other
6
Jabber Server JabberD
Hosts chat room

Figure 57. This diagram shows the flow of data in the initial XTC application. The form is generated on the Web server and displayed by the browser to the client, who fills in the data and submits it to the Web server. The Web server forwards the data to the Jabber server, which posts it in a chat room where a client can read it.

D. SYSTEM DESIGN: INITIAL DEMONSTRATION

To demonstrate XML chat technology four distinct steps are demonstrated: Jabber client login, use of an XHTML form, Web-server services and Jabber-server functionality. Figure 58 presents the steps towards completing these tasks:

1. Jabber Server:
    - Install new tactical chat jabberd server or use an existing one (e.g. xchat.MovesInstitute.org)
    - Receive Jabber messages from the Web server.
    - Strip Jabber tags.
    - Log the message.
    - Put the message in the conference room to be viewed by all clients.
2. Jabber Client
    - Choose an off-the-shelf Jabber client and install it. (The Exodus, RhymBox, and BuddySpace clients were tested.)
    - Configure a Jabber client to connect to a Jabber server.
    - Login into the Jabber server.
    - Join a conference room.
3. XHTML Form
    - Open a Web browser and an XHTML tactical-report form.
    - Fill out the XHTML form and submit it, using the Post button.
        - The Javascript embedded in the Web page does very limited validation of the data the user entered.
        - The Javascript sends the plain text data (strings) to the Web server.
4. Web Server:
    - Receive the text data
    - Build an XML file by inserting the data into a blank XML file
    - Validate the XML file against the proper schema
    - Wrap the original text data in Jabber tags and send to Jabber server
(This causes all clients to see only the text message without XML)
OR
Wrap the XML file in Jabber tags and send to Jabber server
(This causes all clients to see an XML file as text)

Figure 58. Description of initial capabilities that XTC demonstrates, describing roles of Jabber client, XHTML forms, Web server and Jabber server.

Figure 59 shows information flow during an XTC session.

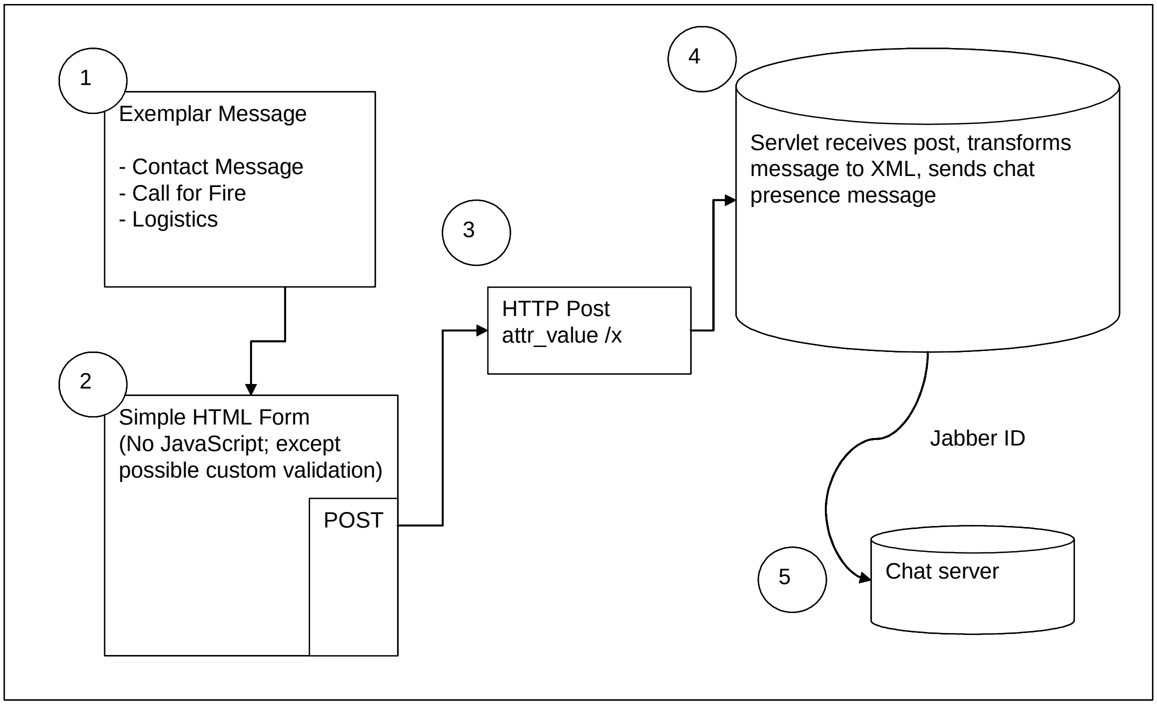

1
Exemplar Message
- Contact Message
- Call for Fire
- Logistics
2
Simple HTML Form
(No JavaScript; except possible custom validation)
POST
3
HTTP Post
attr_value /x
4
Servlet receives post, transforms message to XML, sends chat presence message
Jabber ID
5
Chat server

Figure 59. This diagram shows how an XTC message is sent to the chat server. The user fills out the form and submits it to the Web server using Post. The Web server wraps the message with Jabber tags and logs into the Jabber server with its JID, and sends the message.

1. Jabber Clients

All the Jabber clients evaluated are able to perform basic functions. They can log into a Jabber server, send a message from client to client, send an XML message from client to server, and view XML code in a debugging window.

As discussed in Chapter V, the conferencing capability of BuddySpace is currently not working as documented.

2. XHTML Forms

XHTML forms are in the following message formats for demonstration purposes: contact report, position report, request for fire report, and obstacle report. The forms are filled in by the user and displayed in the browser as an XHTML form, shown in the example layout in Figure 60).

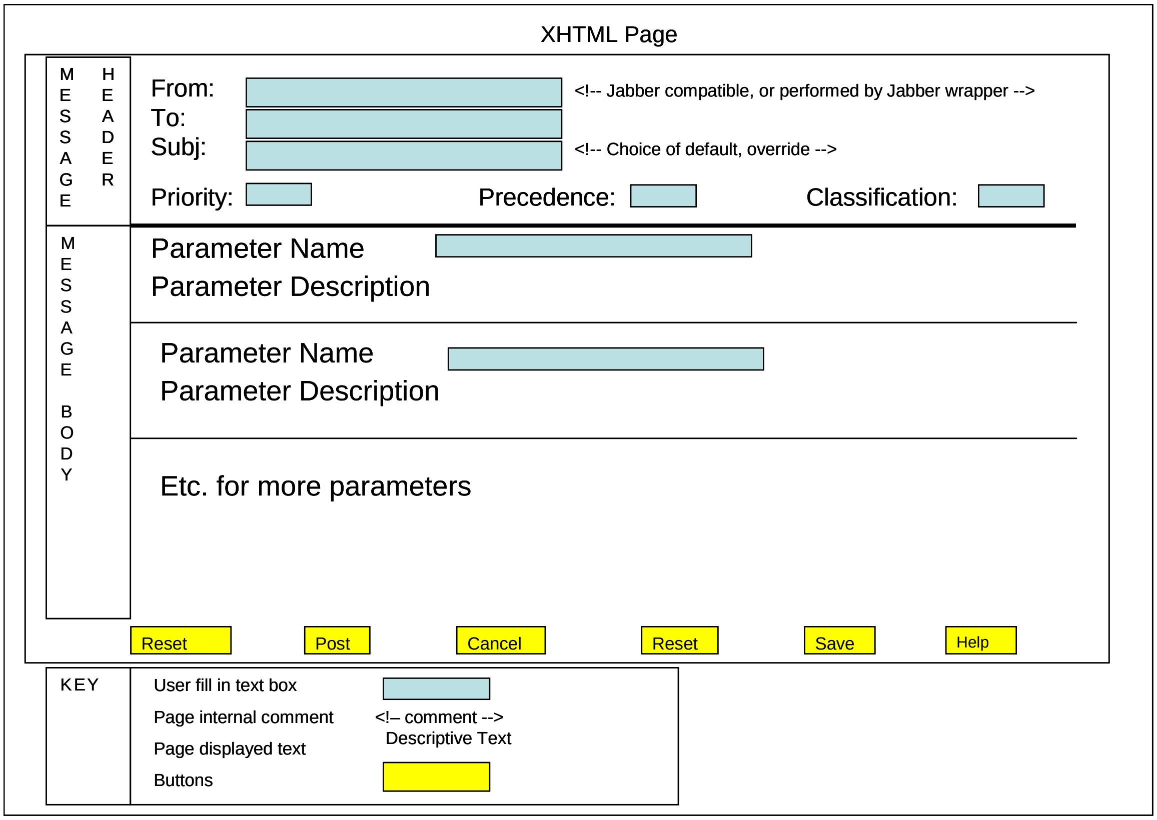


Figure 60. Sample XHTML form for drafting and posting (via server) to a chat channel.

The major benefit of using XHTML forms is that submitting a form is independent of other Jabber clients (P2P/A2A/A2P). In addition, because XHTML forms follow a consistent design pattern, they can be auto-generated from a message schema in the future. Figure 61 shows the flow of XML data from XHTML forms to the Jabber Server:

XHTML form	↓ XSLT
=> XML format for data	↓ HTML
=> POST as to Web Server	POST to Web Server URL
=> Jabber envelop wrapping	↓ servlet, XML
=> send to Jabber Server	send to Jabber URL

Figure 61. The flow of the XML data from the XHTML forms to the Jabber Server in XTC, showing steps involved in using XSLT, HTML and servlets to convert XHTML form into Jabber message.

3. Posting XML to the Server

The Post, rather than the Get, method is used when submitting XML to the Web server so as to get the maximum length possible. The goal is to output XML to the Web server, where a servlet can wrap it and send it to a Jabber server.

HTML POST supports "name-value" pairs only, e.g. `<input type="text" name="edtSubject" value="This is the subject.">` When the servlet receives the HTTP POST request, it sees `"edtSubject"="This is the subject"`, where `"edtSubject"` is the HTML object name, and `"This is the subject"` is the HTML object value.

It is possible to programmatically create XML string (element/attribute/values) on the client Internet browser. To store the XML string value, it needs an HTML object. The norm is to create a hidden object (not visible on client browser). `<input type="hidden" name="hidXMLString" value="Where XML string should be.">` After creation of XML string, assign it to the hidden HTML object. The resulting string looks like this (again "name-value" pair): `"hidXMLString"="<xml>Where XML string should be.</xml>"` To the servlet, it sees the `"name-value"` pair. It searches for HTML object name = `"hidXMLString"` and extracts the HTML object value.

There are other ways to post XML data to the server, e.g., via a Web-services approach with SOAP. However, such techniques are not supported natively by the Internet browser.

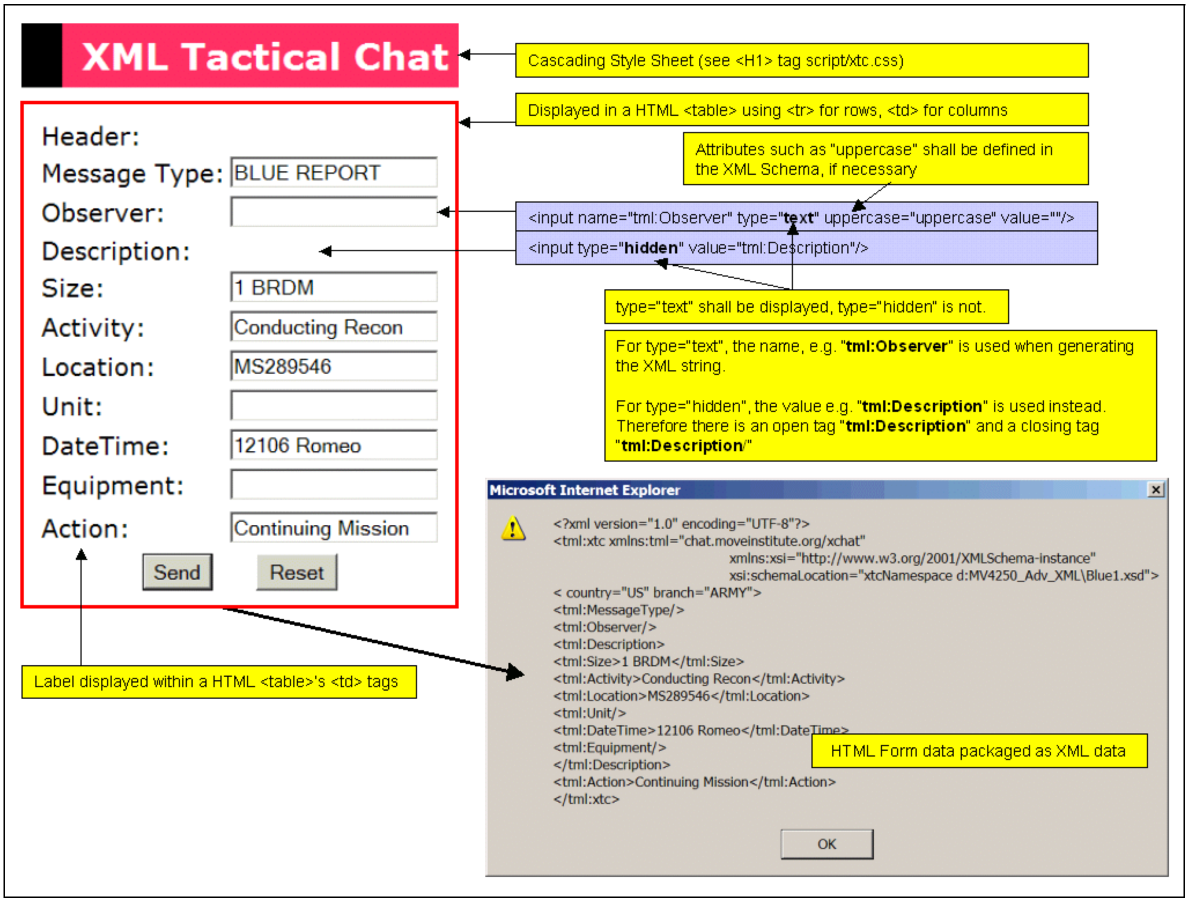
4. Jabber Server

The ideal Jabber server is an off-the-shelf, open-source server that is implemented without modification. The Jabber server can reside on the same machine as the Web server, but is not required to. Jabberd is used for XTC, providing login, presence, conference room, and message-sending capability. Chat messages are pushed to clients or cached until reconnection. Jabberd also provides server-to-server communications and logging.

E. TACTICAL CHAT MESSAGING USER INTERFACE

This section provides screenshots and a discussion of the initial implementation of an auto-generated XTC message-sending webpage client.

Figure 62 shows the XTC form and description boxes for the invisible entries in the form, including default input text and hidden values.



XML Tactical Chat

Header:	
Message Type:	BLUE REPORT
Observer:	
Description:	
Size:	1 BRDM
Activity:	Conducting Recon
Location:	MS289546
Unit:	
DateTime:	12106 Romeo
Equipment:	
Action:	Continuing Mission

Cascading Style Sheet (see <H1> tag script/xtc.css)

Displayed in a HTML <table> using <tr> for rows, <td> for columns

Attributes such as "uppercase" shall be defined in the XML Schema, if necessary

```
<input name="tml:Observer" type="text" uppercase="uppercase" value=""/>
<input type="hidden" value="tml:Description"/>
```

type="text" shall be displayed, type="hidden" is not.

For type="text", the name, e.g. "**tml:Observer**" is used when generating the XML string.

For type="hidden", the value e.g. "**tml:Description**" is used instead. Therefore there is an open tag "**tml:Description**" and a closing tag "**tml:Description/**"

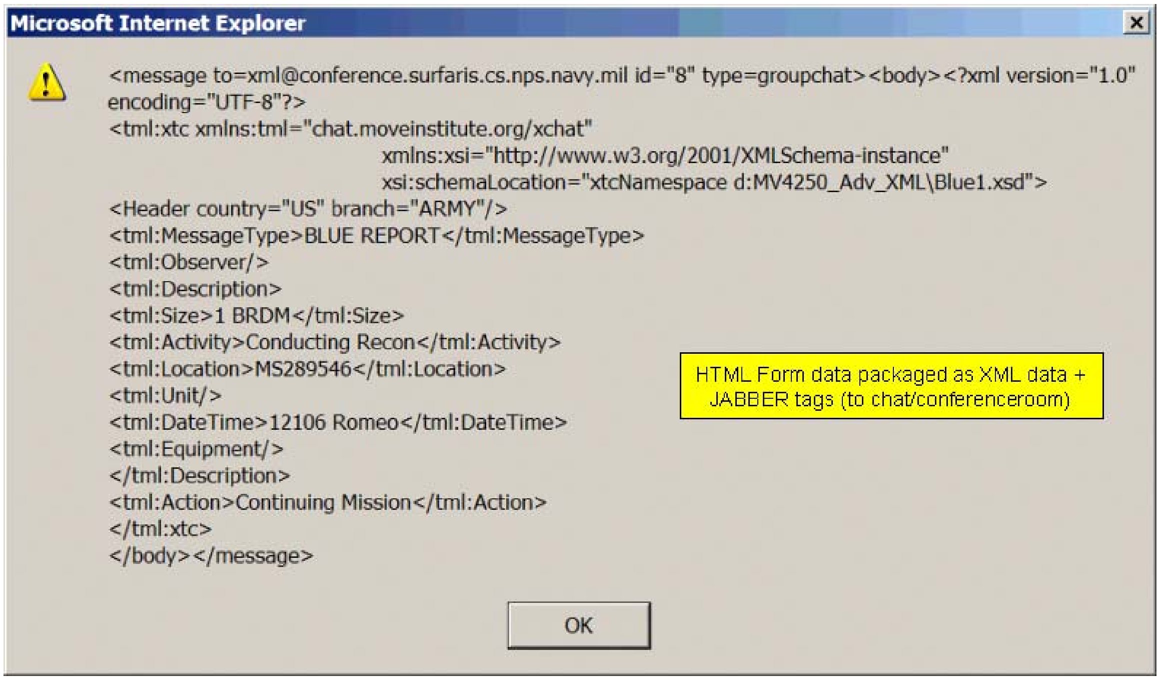
Label displayed within a HTML <table>'s <td> tags

```
<?xml version="1.0" encoding="UTF-8"?>
<tml:xtc xmlns:tml="chat.moveinstitute.org/xchat"
                    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                    xsi:schemaLocation="xtcNamespace d:MV4250_Adv_XML\Blue1.xsd">
< country="US" branch="ARMY">
<tml:MessageType/>
<tml:Observer/>
<tml:Description>
<tml:Size>1 BRDM</tml:Size>
<tml:Activity>Conducting Recon</tml:Activity>
<tml:Location>MS289546</tml:Location>
<tml:Unit/>
<tml:DateTime>12106 Romeo</tml:DateTime>
<tml:Equipment/>
</tml:Description>
<tml:Action>Continuing Mission</tml:Action>
</tml:xtc>
```

HTML Form data packaged as XML data

Figure 62. Shows the XHTML form and the types of XML tags behind it including text and hidden fields.

Figure 63 shows the form data as it is submitted to the Web server.



```
<message to=xml@conference.surfaris.cs.nps.navy.mil id="8" type=groupchat><body><?xml version="1.0"
encoding="UTF-8"?>
<tml:xtc xmlns:tml="chat.moveinstitute.org/xchat"
                                    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                                    xsi:schemaLocation="xtcNamespace d:MV4250_Adv_XML\Blue1.xsd">
<Header country="US" branch="ARMY"/>
<tml:MessageType>BLUE REPORT</tml:MessageType>
<tml:Observer/>
<tml:Description>
<tml:Size>1 BRDM</tml:Size>
<tml:Activity>Conducting Recon</tml:Activity>
<tml:Location>MS289546</tml:Location>
<tml:Unit/>
<tml:DateTime>12106 Romeo</tml:DateTime>
<tml:Equipment/>
</tml:Description>
<tml:Action>Continuing Mission</tml:Action>
</tml:xtc>
</body></message>
```

HTML Form data packaged as XML data + JABBER tags (to chat/conferenceroom)

Figure 63. XHTML Form data wrapped with Jabber tags and ready to send to Jabber server.

F. TACTICAL CHAT MESSAGE STRUCTURES

This section shows how text is put into XML, validated by the schema then sent to the Jabber server. Figure 64 is a plain message as a text file:

```
Alpha: Red 5
Bravo: 1 BRDM
    Conducting Recon
    MS289546
    191206 Romeo
Charlie: Continuing Mission
```

Figure 64. Original Text for a Blue1 report that is entered into the XHTML form.

Each type of report has its own blank form. Figure 65 is the Blue1 report.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xtc
xmlns="http://www.movesinstitute.org/xtc/schemas/TacticalMessageTemplates1.0.xs
d" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.movesinstitute.org/xtc/schemas/TacticalMessageTe
mplates1.0.xsd TacticalMessageTemplates1.0.xsd">
   <TacticalMessage>
      <SpotReport>
         <head messageType="Blue 1" name="Spot or Contact Report" country="US"
branch="Army"/>
         <body>
            <Observer input="true" label="Observer" required="true"/>
            <Description>
               <Size input="true" label="Size" required="true"/>
               <Activity input="true" label="Activity" required="true"/>
               <Location input="true" label="Location" required="true"/>
               <Unit input="true" label="Unit" required="false"/>
               <DateTime input="true" label="Time" required="true"/>
               <Equipment input="true" label="Equipment" required="false"/>
            </Description>
            <Action input="true" label="Observer Action" required="true"/>
         </body>
      </SpotReport>
   </TacticalMessage>
</xtc>
```

Figure 65.	Blank XML Document for Blue1 Report

That plain-text data is typed into the form by the user, which results in the following XML document, Figure 66.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xtc
xmlns="http://www.movesinstitute.org/xtc/schemas/TacticalMessageTemplates1.0.xs
d" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.movesinstitute.org/xtc/schemas/TacticalMessageTe
mplates1.0.xsd TacticalMessageTemplates1.0.xsd">
   <TacticalMessage>
      <SpotReport>
         <head messageType="Blue 1" name="Spot or Contact Report" country="US"
branch="Army"/>
         <body>
            <Observer>Red 5</Observer>
            <Description>
               <Size>1 BRDM</Size>
               <Activity>Conducting Recon</Activity>
               <Location>MS289546</Location>
               <Unit/>
               <DateTime>191206 Romeo</DateTime>
               <Equipment/>
            </Description>
            <Action>Continuing Mission</Action>
         </body>
      </SpotReport>
   </TacticalMessage>
</xtc>
```

Figure 66.	Original Message as an XML Document

The received XML form inputs are then validated by the schema for that type of report. Figure 67 a section of the XTC schema that applies to the Blue1 report. The entire schema can be found in Appendix D.

```
<!-- Blue1 Report Elements (aka Spot Report or Contact Report) -->
   <xs:element name="Observer">
      <xs:complexType mixed="true">
         <xs:attribute name="input" fixed="true"/>
         <xs:attribute name="label" fixed="Observer"/>
         <xs:attribute name="required" fixed="true"/>
      </xs:complexType>
   </xs:element>
   <xs:element name="Size">
      <xs:complexType mixed="true">
         <xs:attribute name="input" fixed="true"/>
         <xs:attribute name="label" fixed="Size"/>
         <xs:attribute name="required" fixed="true"/>
      </xs:complexType>
   </xs:element>
   <xs:element name="Activity">
      <xs:complexType mixed="true">
         <xs:attribute name="input" fixed="true"/>
         <xs:attribute name="label" fixed="Activity"/>
         <xs:attribute name="required" fixed="true"/>
      </xs:complexType>
   </xs:element>
   <xs:element name="Location">
      <xs:complexType mixed="true">
         <xs:attribute name="input" fixed="true"/>
         <xs:attribute name="label" fixed="Location"/>
         <xs:attribute name="required" fixed="true"/>
      </xs:complexType>
   </xs:element>
   <xs:element name="Unit">
      <xs:complexType mixed="true">
         <xs:attribute name="input" fixed="true"/>
         <xs:attribute name="label" fixed="Unit"/>
         <xs:attribute name="required" fixed="false"/>
      </xs:complexType>
   </xs:element>
   <xs:element name="DateTime">
      <xs:complexType mixed="true">
         <xs:attribute name="input" fixed="true"/>
         <xs:attribute name="label" fixed="Time"/>
         <xs:attribute name="required" fixed="true"/>
      </xs:complexType>
   </xs:element>
   <xs:element name="Equipment">
      <xs:complexType mixed="true">
         <xs:attribute name="input" fixed="true"/>
         <xs:attribute name="label" fixed="Equipment"/>
         <xs:attribute name="required" fixed="false"/>
      </xs:complexType>
   </xs:element>
   <xs:element name="Description">
      <xs:complexType>
         <xs:sequence>
            <xs:element ref="xtc:Size"/>
            <xs:element ref="xtc:Activity"/>
            <xs:element ref="xtc:Location"/>
            <xs:element ref="xtc:Unit"/>
            <xs:element ref="xtc:DateTime"/>
            <xs:element ref="xtc:Equipment"/>
         </xs:sequence>
      </xs:complexType>
   </xs:element>
   <xs:element name="Action">
      <xs:complexType mixed="true">
         <xs:attribute name="input" fixed="true"/>
         <xs:attribute name="label" fixed="Observer Action"/>
         <xs:attribute name="required" fixed="true"/>
      </xs:complexType>
   </xs:element>
   <xs:element name="SpotReport">
      <xs:complexType>
         <xs:sequence>
            <xs:element ref="xtc:head"/>
            <xs:element name="body">
               <xs:complexType>
                  <xs:sequence>
                     <xs:element ref="xtc:Observer"/>
                     <xs:element ref="xtc:Description"/>
                     <xs:element ref="xtc:Action"/>
                  </xs:sequence>
               </xs:complexType>
            </xs:element>
         </xs:sequence>
      </xs:complexType>
   </xs:element>
       <!-- End Blue 1 Elements -->
```

Figure 67. The Blue1 section of the XTC schema that validates the XML Document. The entire schema can be found in Appendix D.

Once validated the XML document is ready for the servlet to send it to the Jabber server. Figure 68 shows the format for the validated Blue1 message.

```
<?xml version="1.0" encoding="UTF-8"?>
<tml:xtc xmlns:tml="www.xchat.movesinstitute.org/xchat">
   <tml:Header country="US" branch="Army"/>
   <tml:MessageType>Blue 1</tml:MessageType>
   <tml:Observer>Red 5</tml:Observer>
   <tml:Description>
      <tml:Size>1 BRDM</tml:Size>
      <tml:Activity>Conducting Recon</tml:Activity>
      <tml:Location>MS289546</tml:Location>
      <tml:Unit/>
      <tml:DateTime>191206 Romeo</tml:DateTime>
      <tml:Equipment/>
   </tml:Description>
   <tml:Action>Continuing Mission</tml:Action>
</tml:xtc>
```

Figure 68. Shows the original message after being validated and ready for the servlet to send it to Jabber.

The Jabber tags are then wrapped around the XML document and the message is sent to the Jabber server. Figure 69 shows the valid Jabber tags for a login, P2P, and conference-room message.

“Logon to”

```
<iq id=“BS_AUTH_1” type=“set”>
          <query xmlns=“jabber:iq:auth”>
                    <username>hello</username>
                    <password>world</password>
                    <resource>Surfaris</resource>
          </query>
</iq>
```

“Peer-to-peer” message

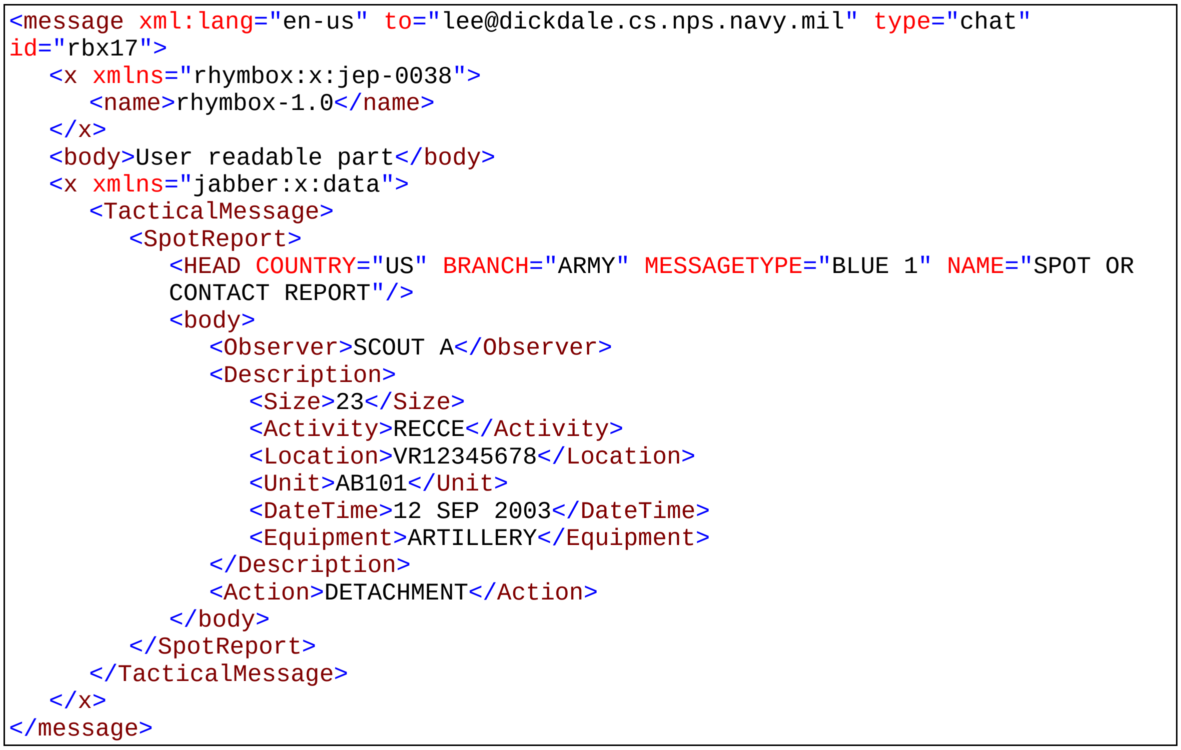
```
<message to=“brutzman@surfaris.cs.nps.navy.mil” id=“3”>
          <subject>Test Subject</subject>
          <body>Test Body</body>
</message>
```

“Conference/Chat room” message

```
<message to=“savage@conference.xchat.movesinstitue.org” id=“8” type=“groupchat”>
          <body>Send to Savage chat room</body>
</message>
```

Figure 69. An example of valid Jabber tags for login, peer to peer (P2P) and conference room messages.

Depending on the message type, Jabbers wrappers are applied. Figures 70 and 71 show the Jabber message format for P2P and chatroom messages.



```
<message xml:lang="en-us" to="lee@dickdale.cs.nps.navy.mil" type="chat"
id="rbx17">
   <x xmlns="rhymbox:x:jep-0038">
      <name>rhymbox-1.0</name>
   </x>
   <body>User readable part</body>
   <x xmlns="jabber:x:data">
      <TacticalMessage>
         <SpotReport>
            <HEAD COUNTRY="US" BRANCH="ARMY" MESSAGETYPE="BLUE 1" NAME="SPOT OR
            CONTACT REPORT"/>
            <body>
               <Observer>SCOUT A</Observer>
               <Description>
                  <Size>23</Size>
                  <Activity>RECCE</Activity>
                  <Location>VR12345678</Location>
                  <Unit>AB101</Unit>
                  <DateTime>12 SEP 2003</DateTime>
                  <Equipment>ARTILLERY</Equipment>
               </Description>
               <Action>DETACHMENT</Action>
            </body>
         </SpotReport>
      </TacticalMessage>
   </x>
</message>
```

Figure 70. Jabber “message sent” format: peer to peer.

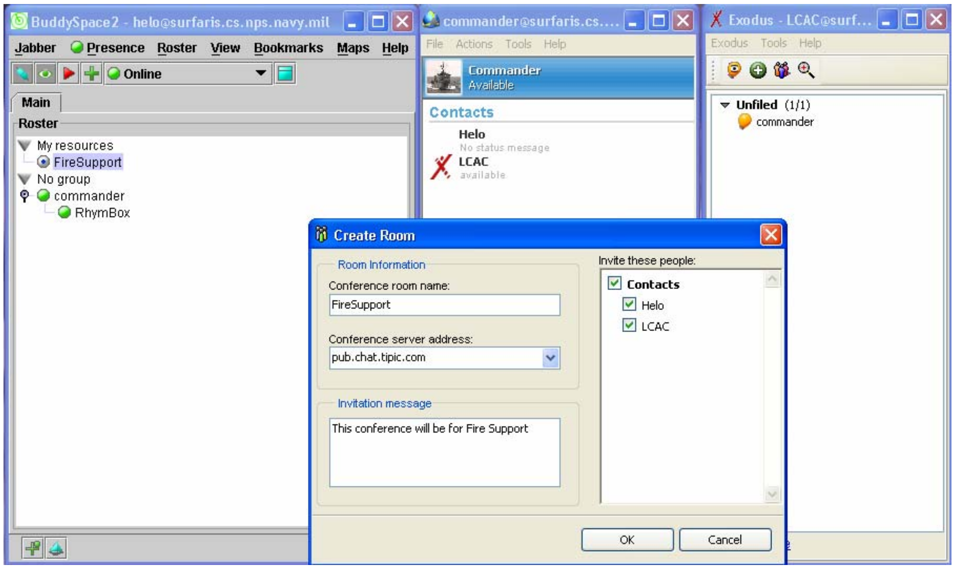
When sending to a conference room, the wrapper is slightly different.

```
<message to="xml@conference.surfaris.cs.nps.navy.mil" type="groupchat">
<body>User readable part</body>
   <x xmlns="jabber:x:data" type="form">
      <TacticalMessage>
         <SpotReport>
            <HEAD COUNTRY="US" BRANCH="ARMY" MESSAGETYPE="BLUE 1" NAME="SPOT OR
            CONTACT REPORT"/>
            <body>
               <Observer>SCOUT A</Observer>
               <Description>
                  <Size>23</Size>
                  <Activity>RECCE</Activity>
                  <Location>VR12345678</Location>
                  <Unit>AB101</Unit>
                  <DateTime>12 SEP 2003</DateTime>
                  <Equipment>ARTILLERY</Equipment>
               </Description>
               <Action>DETACHMENT</Action>
            </body>
         </SpotReport>
      </TacticalMessage>
   </x>
</message>
```

Figure 71. Jabber “message sent” format: chatroom.

G. TACTICAL DEMONSTRATION PLAN

The following scenario demonstrates a notional “call for fire” report during an amphibious assault. There are three participants: a commander, an LCAC and a helo. They will be logged into a conference room using off-the-shelf Jabber clients.



Conference room name: FireSupport

Conference server address: pub.chat.tipic.com

This conference will be for Fire Support

☑ Contacts
☑ Helo
☑ LCAC

Figure 72. Screenshot of commander’s console as he creates a room and invites participants to use XTC.

The LCAC pulls up his XTC "call for fire" form and fills out the information.

All parties see the information at the same time in the chat room.

The commander reviews the request and assigns it to the helicopter.

The helicopter crew then fires.

H. POLICY ISSUES

Firewall policy is one of the biggest hurdles when designing a network protocol. Figure 73 is a diagram showing the current policy at NPS, which is typical of most DoD installations. It shows that HTTP, HTTPS, and SSL ports are available but everything else is blocked.

The policy question is: Can XML-based Jabber chat be sent through a firewall safely?

The fact is XML can already be passed through a firewall as an attachment to an e-mail, and Jabber chat is more secure than e-mail because chat clients don't execute externally provided code. In addition, Jabber is set up so that servers, not clients, talk to each other. Thus the only connection needed is from a server outside to a server inside.

A server-centered S2S approach minimizes the risk of chat through a firewall because:

- There are only single points of failure.
- It is not prone to abuse (for example, spam).
- Traffic on the server is in plain XML and can be easily monitored.

The Jabber protocol is an XML stream over a TCP socket. In support of common firewall architecture, it allows standard SOCKS and IP Masquerading. [Support 02].

The current firewall policies regarding XML-based Jabber at NPS are:

- http:// [port 80] for regular web-server access is open for outgoing requests.

- https:// [port 443] for secure web-server access is open for outgoing requests.
- Jabber [port 5222] for Jabber client to server communication is blocked.
- Jabber [port 5223] for Jabber client to server SSL communication is blocked.
- Jabber [port 5269] for Jabber server to server communication is open between surfaris and xchat servers only.

When configuring firewall settings, web-server proxy polling access can be considered if other firewall ports unavailable. Appendix E covers Navy policy. Figure 73 is a diagram of the current port configurations at NPS.

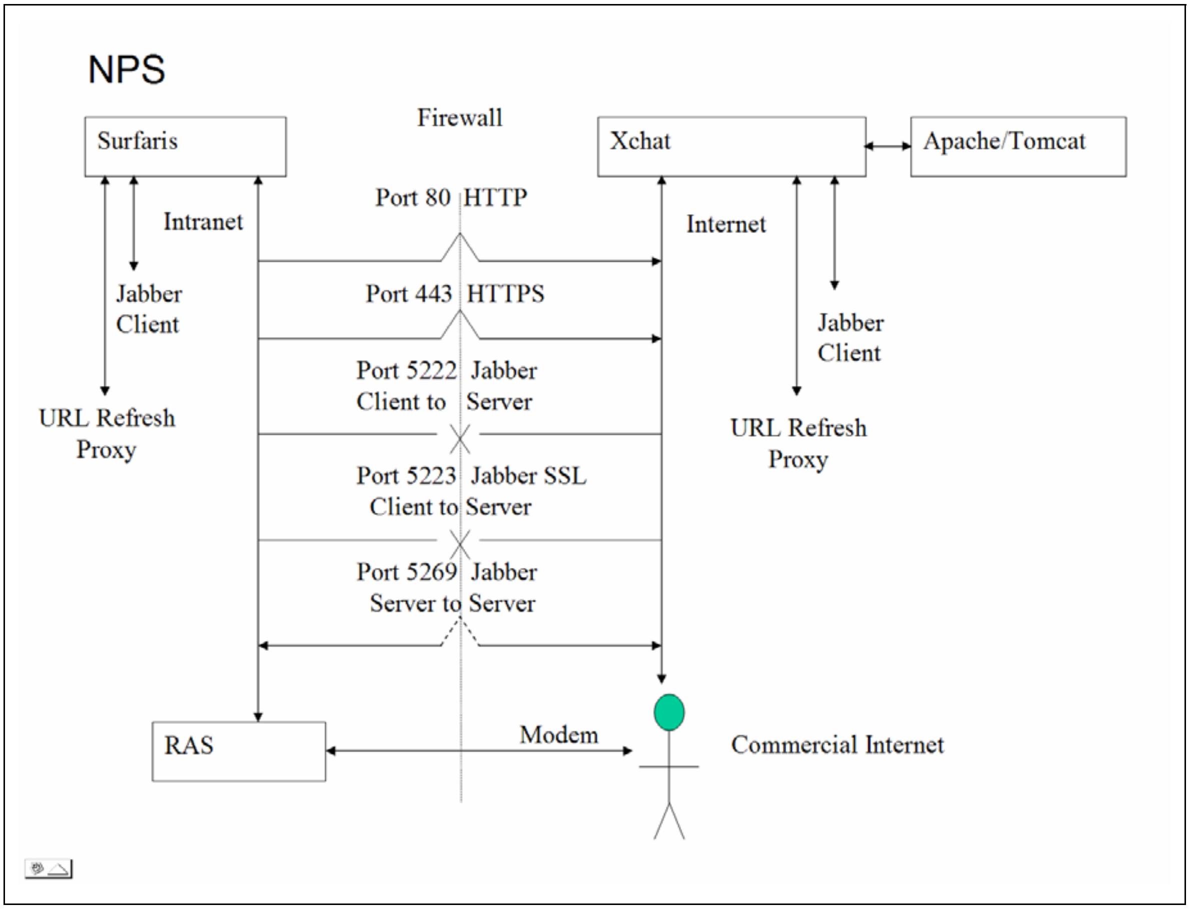


Figure 73. Current communications diagram and available ports at NPS. Ports 5222 and 5223 for client to server Jabber communications are disabled and port 5269 for server to server Jabber communications is only enabled between surfaris and xchat.

I. RELATED DEVELOPMENTS

This section describes the lessons learned from implementing XTC and changes in Jabber that will directly impact XTC.

1. Lessons Learned

- (Server) To set up a conference (chat room) a separate DNS entry is needed.
- (Client) Jabber User Id (JID) is not the same as the nickname displayed in the client. I.e., names are not eponymous. JID is required to send Jabber message.

- (Client) Exodus saves chat as in RTF, not XML.
- When using Jabber, typical account names are of the form "username@server" and conference rooms are "conferenceroomname@conference.server."
- User accounts are automatically added when connecting and the account doesn't exist.
- Jabber can be linked to legacy services (e.g. Yahoo, MSN, AIM, ICQ).

2. Further Changes In Jabber

Internet2, a consortium of over 200 universities that is working with government and industry to define the next-generation Internet, has formed an instant-messaging workgroup that will standardize on Jabber/XMPP technology for campus and inter-campus IM networks. URL: http://middleware.internet2.edu/i2im.

The replacement to Jabberbeans is as follows: http://www.jabberstudio.org/projects/b9j2/project/view.php

An alternative open-source chat environment is Babylon chat, written in Java with whiteboard functionality http://www.visopsys.org/andy/babylon/index.html.

J. SUMMARY

This chapter shows how an initial XTC application was implemented using jabberd, XHTML forms, servlets, and off-the-shelf Jabber clients. It describes the fundamental technologies of XML, XHTML, and open-source software, and how data from an XHTML form is sent to a Jabber server. Finally, policy issues and XTC-related developments are briefly discussed.

VII. SECURITY CONSIDERATIONS

A. INTRODUCTION

This chapter presents a simple overview of security considerations when using chat, and the importance of code reliability, information assurance, and NMCI/IT-21 compatibility. The first section summarizes the security requirements for chat. Next, the importance and value of code reliability inherent in open-source software is described. Information Assurance (IA) ensuring the privacy of chat communications is then discussed. The last section describes how XTC might become compliant with Task Force Web (TFW) specifications.

B. SECURITY REQUIREMENTS

There are three main security considerations for chat, listed in Figure 74.

- Code reliability
- Information assurance (IA)
- NMCI/IT-21 compatibility

Figure 74. Security requirements for chat.

Code reliability means that the code performs as expected, without backdoors or Trojan horses. Information assurance (IA) guarantees that data reaches its intended recipient and no one else. Meeting NMCI requirements will permit the use of XTC on Navy networks.

C. CODE RELIABILITY – OPEN SOURCE

The expectation that software runs as advertised without backdoors or Trojan horses is an important, but often unrecognized aspect, of code reliability. Unfortunately modern programs contain millions of lines of code, perhaps hidden by licensing restrictions, which preclude broad and thorough evaluation of every line. A worthy challenge is to design a process by which all lines of code can be scrutinized.

A major value of open source is that all code is subject to inspection and protected by a select meritocracy of code committers. It is reviewed by the open-source community, which prevents programmers from inserting Trojan horses or backdoor vulnerabilities. A caveat regarding back doors in a non-open source project was given when Ken Thompson, the creator of Unix, revealed that a previous unsuspected back-door vulnerability lurked in every machine running Unix [ACM 95]. These are broad and important issues. Producing open-source XTC software has critical reliability considerations.

D. INFORMATION ASSURANCE – ENSURING CHAT PRIVACY

To ensure that chat messages are secure and unreadable by third parties, the data coming to and from the chat client must be encrypted, best accomplished by implementing SSL, which is available for Jabber, but not to users of AIM, ICQ, MSN, and Yahoo.

A good example of the information that can be gained from eavesdropping on chat is evident in the open source application called AIM Sniff, available at http://www.aimsniff.com. With this tool users can snoop on anyone using AOL Instant Messenger. Figure 75 lists several capabilities of AIM Sniff.

- Monitor AIM messages, either through actively sniffing the network, or by reading a “packet capture library” file
- Archive messages
- Export messages to a MySQL database, flat file, STDOUT, or any combination of the three
- Monitor how often AIM is used
- Check for file swapping

Figure 75. Security vulnerabilities of AIM as exposed by AIM Sniff [Rose 03].

E. NMCI/IT-21 REQUIREMENTS FROM TFW

Task Force Web security requirements are published in the *Application Development Guide, Appendix G: Application Security*, provided for convenient reference in Appendix F. According to these requirements, a secure (future-version) of XTC is responsible for securing the user-facing service (UFS) interface. A possible solution for an XTC custom client is implementing a portal-supplied common identity with SSL. This can ensure a sufficient level of trust while enabling the interoperability features inherent in having a portal-common identity. The goal of XTC is not to reinvent this process, rather incorporate the methods used by TFW to ensure future interoperability DoD wide. This task is important future work.

F. SUMMARY

The chapter discussed security considerations incorporated into XTC and the importance of code reliability, information assurance, and NMCI/IT-21 compatibility. Open-source programming ensures that code reliability is achieved. Information assurance (IA) requirements using SSL guarantee that data sent across chat is secure. NMCI/IT-21 compatibility enables XTC implementation into the TFW portal. Future work on chat and XTC security is vitally important.

THIS PAGE INTENTIONALLY LEFT BLANK

VIII. CONCLUSIONS AND FUTURE WORK

A. INTRODUCTION

This chapter summarizes the evaluated ability of XTC to meet all the requirements for tactical chat laid out in Chapter III. Current success and failures of XTC are first presented as summary conclusions, followed by recommendations for future work to continue building up successful capabilities of XTC.

B. CONCLUSIONS

The following conclusions show how XTC answers the questions posed at the beginning of this report.

- *What benefits can XML-based tactical chat provide to military communications?*
  - Flexiblity – can adapt for the specific needs of the military
  - Universality – works on many different system
  - Extensiblity – can grow to incorporate additional features/capabilities
  - Security – supports SSL
  - Storablity – All data is in XML
- *What are the tactical requirements for XML-based tactical chat?*

  XTC met all the tactical requirements listed earlier, specifically:

  - Support of command and control, to include open dialog as well as situation reports, execution checklist milestones and casualty reports
  - Support of operational planning at the micro- and macro levels for both future and real-time event scheduling and coordination
  - Support of coordination efforts for administrative support, logistics, technical support, and other day-to-day requirements
  - Chat-logging so that valuable information is preserved
  - Ability to XML-ize chat to facilitate extraction of valuable information

- *What are the technical requirements for XML-based tactical chat?*

  XTC met the majority of technical requirements, with a few pending. The requirements met are:

  - Mark up using standardized XML
  - Processing of plain prose, MTF, BGH TML, HTML, and BGH ATTCS reference model messages
  - Message validation against a schema
  - BGH element compatibility with the BGH architecture.
  - Asynchronicity
  - Client thinness
  - Interoperablity
  - Firewall policy compatiblity
  - Employment of open source

- *What are the administrative requirements for XML-based tactical chat?*

The solution used for XTC was off the shelf and not customized. The administrative requirements including defining user roles, user permissions, scheduling, and DITSCAP compliance are not contained in the off the shelf clients. All administrative requirements are represented in the future work section.

- *How can XML be applied to chat?*

  The data from chat can be stored in XML thus easily stored, searched and analyzed.

- *What is Jabber and how can it be used for tactical chat?*

  Jabber is a new, open-source, chat protocol that enables asynchronous XML formatted conversation. It is reliable, secure, and can be extended and customized to meet tactical-chat functionality.

- *What open-source Jabber clients are available and how are they implemented?*

  We evaluated three open-source Jabber clients: BuddySpace, Exodus, and Rhymbox. These clients are in different stages of development and each has desirable features. Because BuddySpace conferencing capability does not currently work with our server, it cannot be used with XTC at this time.

- *Can XML be sent from a client to a Jabber server and displayed?*

  By using the debugging window on the Jabber client, the XML between the servers and the clients can be viewed.

- *Can chat information be sent using a thin client (XHTML page) instead of thick (Jabber client)?*

  Yes, it was demonstrate that XML data can be sent from XHTML, validated, sent to the Jabber server, and displayed.

- *Can XML-based chat also be used for software-agent and combat-control systems (CCS) communications?*

  Yes, proving that a servlet can communicate with the Jabber server XTC shows that software-agent communications are possible.

- *How is a Jabber server configured?*

  This question is expanded because XTC implements a Web server/Java server solution. The Jabber server is configured using the jabber.xml file, while the Web server is configured using web.xml, and server.xml.

- *What are the security benefits and concerns when using Jabber?*

The main security considerations for chat code are reliability, information assurance, and NMCI/IT-21 compatibility. Code reliability ensures that the code is doing what it supposed to and does not contain any back doors or Trojan horses. Information assurance guarantees that the data sent gets to its intended recipient and no one else. By meeting NMCI requirements it will enable the use of XTC on Navy networks.

C. RECOMMENDATIONS FOR FUTURE WORK

1. Issues

The following XML challenges still need addressing through further work:

- How is XSLT best invoked within a browser?
- Can XSLT scripts be composed and chained?
- Can a Jabber server be posted to without prior registration, perhaps bundling all info in a single post or including some framing external page to register first?

The following Jabber questions still remain:

- How can Jabber servers be synchronize? One potential solution is to use synchronizing together with NTP for logging time.
- What is the best way to log messages on the Jabber server? A possible solution that is the Bandersnatch server plug-in to help with logging. http://www.jabberstudio.org/projects/bandersnatch/project/view.php

2. Future Work

The four categories of future work are client and server development, network optimization, and XTC applications, presented in Figures 83-86.

Client development:

- Custom client or fully functional HTML interface to Jabber
  - Log on to Jabber server and check a URL for available message formats
  - Download schema of reports being created and validate on the client
  - Send validated XML to the Jabber server
  - Receive XML formatted reports from the server and display them as text
- Evaluate Web-based Jabber clients like JWChat https://sourceforge.net/projects/jwchat.
- Develop XTC using a Soap Envelope
- Auto install
- Multilingual
- Certify client for DoD systems with an SSAA.

Figure 76. Future client-development work

Server development:

- A server GUI needs to be researched.
- SSL needs to be configured on all servers.
- Research creating a gateway from e-mail to Jabber.
- New server options (jabberd currently a single point of failure) must be found.
- Implement components that bridge other chat clients and SMS.
- Enable user directories
- Maintain single user between two servers.
- Certify server for DoD systems with an SSAA.

Figure 77. Future server-development work

Network transport routing and optimization will be one of the major follow-on challenges for XTC.

- Payload size comparison – Jabber, AOL, MSN, Yahoo, IRC, and XFSP compressed Jabber.
- Long-term network-optimization strategies over diverse DoD links XTC Applications.
- Compare with Envoke system performance.

Figure 78. Future network-optimization work

The last area of work is applying XTC to the tactical environment and designing the desired customized usability functions.

- Use Java access cards for automating watch turnover reports
- In order to support scalable (point-to-multipoint) audio and video, the core XMPP protocol would have to be modified to support secure data (via SSL) [Boyd 03].
- Allow MIME types and URLs to be accepted and stored.
- Incorporate data mining support into the XTC application.
- Define a schema for audio, video, and whiteboarding.
- Allow XSLT templates for available messages to be added, deleted, and modified.
- 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 interest for chat.

Figure 79. Future XTC-usability functional work

D. TRANSFORMATION OPPORTUNITY

Chat is already changing the way joint operations are conducted in the military. XML-based tactical chat (XTC) is able to meet current chat needs in a military environment, overcoming boundaries and limitations faced by current chat tools. The Jabber instant messaging protocols, standards, and software are the clear choice for open interoperability. If fully implemented, the extensible nature of XTC will enable it to incorporate various human and system communications such as e-mail and message traffic, integrating these with system information to potentially reduce user information overload. Thus XTC has the potential to revolutionize tactical communications.

Further work on this critical capability is possible in the near future, highly recommended, and will further enable the transformation of joint coordinated operations.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX A. ACRONYMS

A2A	Application-to-Application
A2P	Application-to-Peer
AIM	America-Online Instant Messanger
ARM	ATCCIS Reference Model
ATCCIS	Army Tactical Command and Control Information System
BGH	Battlespace Generic Hub
BGHTML	Battlespace Generic Hub HTML
CA	Certificate Authorities
CAC	Common Access Card
DAA	Designated Approving Authority
DCTS	Defense Collaboration Tool Suite
DoD	Department of Defense
DIS	Distributed Interactive Simulation
DLL	Dynamic Link Library
DVS	Distributed Virtual Simulations
FBE	Fleet Battle Experiment
FNMOC	Fleet Numerical, Meteorological, and Oceanographic Center at https://www.fnmoc.navy.mil
GMU	George Mason University
HTML	Hypertext Markup Language at http://www.w3.org/MarkUp
HTTP	Hypertext Transfer Protocol at http://www.w3.org/Protocols
HTTPS	Secure http
ICQ	I Seek You
IM	Instant Messaging
IRC	Internet Relay Chat
IT-21	Information Technology 21st century
IETF	Internet Engineering Task Force
IWS	Info Work Space
JID	Jabber ID
JFCOM	Joint Forces Command at http://www.jfcom.mil
JSF	Jabber Software Foundation
LSVE	Large-Scale Virtual Environments
MOE	Measure of Effectiveness
NCES	Network Centric Enterprise Services
NEP	Navy Enterprise Portal at https://portal.tfw.navy.mil
NKO	Navy Knowledge Online at https://wwwa.nko.navy.mil
NMCI	Navy Marine Corps Intranet
NPS	Naval Postgraduate School at http://www.nps.navy.mil
NTP	Network Time Protocol
NUWC	Naval Undersea Warfare Center
ODU	Old Dominion University
OTH	Over-the-Horizon
P2P	Peer to Peer

PGP/GPG	Pretty Good Privacy/Gnu Privacy Guard
PRI	Portal Request Interface
RFC	Request for Comments
S2S	Server to Server
SAR	Search and Rescue
SASL	Simple Authentication Security Layer
SITREP	Situation Report
SMTP	Simple Mail Transfer Protocol
SPAWAR	Space and Naval Warfare Systems Command
SRTP	Selectively Reliable Transport Protocol
SSL	Secure Sockets Layer
TCP	Transmission Control Protocol
TML	Tactical Markup Language
TFWeb	Task Force Web
VRML 97	Virtual Reality Modeling Language
UFS	User Facing Service
USMTF	United States Message Text Format
USJFCOM	United States Joint Forces Command
W3C	World Wide Web Consortium at http://www.w3.org
XForms	XML Forms
XHTML	Extensible Hypertext Markup Language
XML	Extensible Markup Language at http://www.w3.org/XML
XMPP	Extensible Messaging and Presence Protocol at http://www.ietf.org/html.charters/xmpp-charter.html
XMSF	Extensible Modeling and Simulation Framework at http://www.movesinstitute.org/xmsf/
XSL	Extensible Stylesheet Language
XSLT	Extensible Stylesheet Language for Transformations
XTC	XML-based Tactical Chat

APPENDIX B. JABBER CONFIGURATION FILE (JABBER.XML)

This is the Jabber server configuration file. The file is broken into different sections based on the services being managed by jabberd, the server daemon. Most of the important sections have comments and are easy to modify. Further instructions including an annotated version of this configuration file and an installation guide are found at http://jabberd.jabberstudio.org/1.4.

```
<jabber>
       <!--
       jabber.xml for xchat.MovesInstitute.org
       changes:
       10 DEC 2003   brutzman      modified rlogger format
       -->
       <!--
       Note that when you see a tag like "jabberd:cmdline", it's automatically
replaced on startup with the command line flag passed in to jabberd. This
enables you to override parameters set in this configuration file if necessary
or desired. Also note as you comment things in and out that jabberd does not
like comments within comments, so be careful with your XML. :)
       -->
       <!--
       The following <service/> section is for the session manager, the most
important component within the server. This section contains the following
types of information:
       * the server's hostname
       * other basic server information
       * the location of the session log file
       * email addresses for server administrators
       * registration instructions for new users
       * a welcome message for new users
       * a list of agents with which users can register
       * load rules for the modules within the session manager
       -->
       <service id="sessions">
       <!–
       Replace all occurrences of "localhost" in this file by the hostname of
your Jabber server. Be aware changing the server's name is all but impossible
once users start to use the server. So choose a name that is permanent
(especially no Intranet hostnames or IP addresses).
       Multiple <host/> entries are allowed - each one is for a separate virtual
server. Note that each host entry must be on one line, the server doesn't like
it otherwise! :)
       Use lowercase for the hostname.
       -->
              <host>
                     <jabberd:cmdlineflag="h">xchat.movesinstitute.org
                     </jabberd:cmdline>
              </host>
       <!–
       This is the custom configuration section for the Jabber session manager,
a.k.a. "JSM".
       -->
              <jsm xmlns="jabber:config:jsm">
       <!–
```

```
        The <filter/> section below determines settings for mod_filter, a server-
side module built into JSM that enables users to set delivery rules for
messages they receive (not yet supported by all clients). The <allow/>
subsection specifies which conditions and actions to enable. High-level
descriptions of each setting can be found below:
        * <default/> - a user cannot delete this one, it's the default rule for
delivering messages
        * <max_size/> - the maximum number of rules in a user's rule set (we
don't want to overdo it!)
        * conditions...
        * <ns/> - matches the query xmlns attrib on an iq packet
        * <unavailable/> - matches when user is unavailable
        * <from/> - matches the sender of the message
        * <resource/> - matches the receiver's resource
        * <subject/> - matches the subject of the message
        * <body/> - matches the body of the message
        * <show/> - matches the show tag on the receiver's presence
        * <type/> - matches the type of the message
        * <roster/> - matches if the sender is in your roster
        * <group/> - matches if the sender is in the specified group
        * actions...
        * <error/> - replies with an error
        * <offline/> - stores the messages offline
        * <forward/> - forwards the message to another jid
        * <reply/> - sends a reply to the sender of the message
        * <continue/> - continues processing of the rules
        * <settype/> - changes the type of the message
        -->
                        <filter>
                                <default/>
                                <max_size>100</max_size>
                                <allow>
                                        <conditions>
                                                <ns/>
        <!-- Matches if the iq's xmlns is the same as the specified namespace -->
                                                <unavailable/>
        <!-- Flag that matches when the reciever is unavailable (offline) -->
                                                <from/>
        <!-- Matches if the sender's jid is the specified jid -->
                                                <resource/>
        <!-- Matches if the sender's resource (anything after the / in a jid) is
the specified resource -->
                                                <subject/>
        <!-- Matches if the message's subject is the specified subject (no regex
yet) -->
                                                <body/>
        <!-- Matches if the message body is the specified body (no regex yet) -->
                                                <show/>
        <!-- Matches if the receiver's presence has a show tag that is the same
as the specified text -->
                                                <type/>
        <!-- Matches if the type of the message is the same as the specified text
("normal" is okay) -->
                                                <roster/>
        <!-- Flag that matches when the sender is in the receiver's roster -->
                                                <group/>
        <!-- Matches when the sender is in the specified group -->
                                        </conditions>
                                        <actions>
                                                <error/>
        <!-- Sends back an error message to the sender, with the specified text -
->
                                                <offline/>
```

```
    <!-- Flag that stores the message offline -->
                        <forward/>
    <!-- forwards the message to the specified jid -->
                        <reply/>
    <!-- Sends back a reply to the sender with the specified text in the body
-->
                        <continue/>
    <!-- Flag that continues rule matching, after a rule matches -->
                        <settype/>
    <!-- Changes the type of message to the specified type, before delivery
to the receiver -->
                    </actions>
                </allow>
            </filter>
    <!-- The server vCard -->
            <vCard>
                <FN>Jabber Server</FN>
                <DESC>A Jabber Server!</DESC>
                <URL>http://xchat.movesinstitute.org</URL>
            </vCard>
    <!–
    Registration instructions and required fields. The notify attribute will
send the server administrator(s) a message after each valid registration if the
notify attribute is present.
    -->
            <register notify="yes">
                <instructions>Choose a username and password to
register with this server.</instructions>
                <name/>
                <email/>
            </register>
    <!--
    A welcome note that is sent to every new user who registers with your
server. Comment it out to disable this function.
    -->
            <welcome>
                <subject>Welcome!</subject>
                <body>Welcome to the Jabber server at xchat -- we
hope you enjoy this service! For information about how to use Jabber, visit the
Jabber User's Guide at http://jabbermanual.jabberstudio.org/</body>
            </welcome>
    <!--
    IDs with admin access - these people will receive admin
       messages (any message to="yourhostname" is an admin
       message). These addresses must be local ids, they cannot
       be remote addresses.

       Note that they can also send announcements to all
       users of the server, or to all online users. To use
       the announcement feature, you need to send raw xml and be
       logged in as one of the admin users. Here is the syntax
       for sending an announcement to online users:

        <message to="yourhostname/announce/online">
         <body>announcement here</body>
        </message>

        <message to="yourhostname/announce/motd">
         <body>message (of the day) that is sent only once to all users that
are logged in and additionally to new ones as they log in</body>
        </message>

       Sending to /announce/motd/delete will remove any existing
```

```
          motd, and to /announce/motd/update will only update the motd
          without re-announcing to all logged in users.

          The <reply> will be the message that is automatically
          sent in response to any admin messages.
          -->
                           <!--
          <admin>
           <read>support@localhost</read>
           <write>admin@localhost</write>
           <reply>
            <subject>Auto Reply</subject>
            <body>This is a special administrative address. Your message was
received and forwarded to server administrators.</body>
           </reply>
          </admin>
          -->
                           <!--
          This enables the server to automatically update the
          user directory when a vcard is edited. The update is
          only sent to the first listed jud service below. It is
          safe to remove this flag if you do not want any users
          automatically added to the directory.
          -->
                           <vcard2jud/>
                           <!--
          The <browse/> section identifies the transports and other
          services that are available from this server. Note that each
          entity identified here must exist elsewhere or be further
          defined in its own <service/> section below. These services
          will appear in the user interface of Jabber clients that
          connect to your server.
          The <browse/> section is also used by mod_disco (see below)
          for building the disco#items reply.
          -->
                           <browse>
                                 <!--
           This is the default agent for the master Jabber User
           Directory, a.k.a. "JUD", which is located at jabber.org.
           You can add separate <service/> sections for additional
           directories, e.g., one for a company intranet.
           -->
                                 <service type="jud" jid="users.jabber.org"
name="Jabber User Directory">
                                        <ns>jabber:iq:search</ns>
                                        <ns>jabber:iq:register</ns>
                                 </service>
                                 <item category="conference" type="public"
jid="conference.xchat.movesinstitute.org" name="Public Conferencing"
version="0.6.0">
                                        <ns>http://jabber.org/protocol/muc</ns>
                                 </item>
                                 <!--
           The following services are examples only, you will need to
           create/modify them to get them working on your Jabber
           server. See the README files for each service and/or the
           server howto for further information/instructions.
           -->
                                 <!-- we're commenting these out, of course :)

           <service type="aim" jid="aim.localhost" name="AIM Transport">
            <ns>jabber:iq:gateway</ns>
            <ns>jabber:iq:register</ns>
           </service>

           <service type="yahoo" jid="yahoo.localhost" name="Yahoo! Transport">
            <ns>jabber:iq:gateway</ns>
            <ns>jabber:iq:register</ns>
           </service>

           end of <service/> examples -->
                           </browse>
                           <!--
          "Service Discovery" (disco, JEP-0030) supersedes
          "Jabber Browsing" (JEP-0011).
          The <disco/> section is used for building the disco#info reply.
          -->
                           <disco>
                                 <identity category="services" type="jabber"
name="Jabber 1.4 Server"/>
                                 <feature var="jabber:iq:browse"/>
                                 <feature var="jabber:iq:agents"/>
                                 <feature var="jabber:iq:register"/>
                                 <feature var="jabber:iq:time"/>
                                 <feature var="jabber:iq:last"/>
                                 <feature var="jabber:iq:version"/>
                           </disco>
                           <!--
          Select the hashing algorithm that mod_auth_crypt uses
          for storing passwords
          Possible values:
          crypt ... traditional hashing as implemented in crypt()
          SHA1 ... using SHA1 hashes
          -->
                           <mod_auth_crypt>
                                 <hash>SHA1</hash>
                           </mod_auth_crypt>
                           <!--
          Configuration for mod_version. By defining <no_os_version/>
          mod_version will not report the version of your OS.
          -->
                           <!--
          <mod_version>
           <no_os_version/>
          </mod_version>
          -->
                    </jsm>
                    <!--
         The following section dynamically loads the individual
         modules that make up the session manager. Remove or
         comment out modules to disable them. Note that the order
         of modules is important, since packets are delivered
         based on the following order!!
         -->
                    <load main="jsm">
                           <jsm>./jsm/jsm.so</jsm>
                           <mod_echo>./jsm/jsm.so</mod_echo>
                           <mod_roster>./jsm/jsm.so</mod_roster>
                           <mod_time>./jsm/jsm.so</mod_time>
                           <mod_vcard>./jsm/jsm.so</mod_vcard>
                           <mod_last>./jsm/jsm.so</mod_last>
                           <mod_version>./jsm/jsm.so</mod_version>
                           <mod_announce>./jsm/jsm.so</mod_announce>
                           <mod_agents>./jsm/jsm.so</mod_agents>
                           <mod_browse>./jsm/jsm.so</mod_browse>
                           <mod_disco>./jsm/jsm.so</mod_disco>
                           <mod_admin>./jsm/jsm.so</mod_admin>
                           <mod_filter>./jsm/jsm.so</mod_filter>
                           <mod_offline>./jsm/jsm.so</mod_offline>
                           <mod_presence>./jsm/jsm.so</mod_presence>
                           <!--
          Authentication
          For standard setups mod_auth_digest is recommended. Additionally
          enable mod_auth_plain if you need plaintext authentication.
          For maximum security, force SSL connections and use mod_auth_crypt
          exclusively. Be aware encrypted password storage can lead to
          problems when migrating to other authentication mechanisms
          (LDAP...).
          Switching from plain/digest to crypt needs manual work for
          existing accounts, the reverse is not possible.
          http://jabberd.jabberstudio.org/1.4/doc/adminguide#security
          -->
                           <!-- mod_auth_digest: Password in clear text in
storage,
             encrypted/hashed on the wire -->
                           <mod_auth_digest>./jsm/jsm.so</mod_auth_digest>
                           <!-- mod_auth_plain: Password in clear text in
storage
             and on the wire. Disable this if you do not use clients
             that need plaintext auth -->
                           <mod_auth_plain>./jsm/jsm.so</mod_auth_plain>
                           <!-- mod_auth_crypt: Password encrypted/hashed in
storage,
             clear text on the wire. Disabled as this only makes
             sense when used exclusively and with SSL mandatory
          <mod_auth_crypt>./jsm/jsm.so</mod_auth_crypt> -->
                           <mod_log>./jsm/jsm.so</mod_log>
                           <mod_register>./jsm/jsm.so</mod_register>
                           <mod_xml>./jsm/jsm.so</mod_xml>
                    </load>
             </service>
             <!-- OK, we've finished defining the Jabber Session Manager. -->
             <!--
        The <xdb/> component handles all data storage, using the filesystem.
        Make sure the spool directory defined here exists and has proper
        permissions.
        -->
             <xdb id="xdb">
                    <host/>
                    <load>
                           <xdb_file>./xdb_file/xdb_file.so</xdb_file>
                    </load>
                    <xdb_file xmlns="jabber:config:xdb_file">
                           <spool>
                                  <jabberd:cmdline
flag="s">./spool</jabberd:cmdline>
                           </spool>
                    </xdb_file>
             </xdb>
             <!--
        The following service manages incoming client socket connections.
        There are several items you can set here to optimize performance:

         * authtime - default is unlimited, but you can set this to
          limit the amount of time allowed for authentication to be
          completed, e.g., <authtime>10</authtime> for 10 seconds

         * heartbeat - default is to not send out heartbeat packets
          to the clients. This option allows you to specify that
```

```
   you want heartbeats to happen every x seconds. This is
   useful if you have a lot of dial-up or laptop users who
   may drop their connection without logging off of jabber.
   Otherwise the server won't notice that they are offline until
   someone tries to send a packet to them (and the message is
   lost). Example: <heartbeat>60</heartbeat>

  * karma - this is an input/output rate limiting system that
   the Jabber team came up with to prevent bandwidth hogging.
   For details about karma, read the io section at the bottom.
   These are the low settings and apply per connection/socket
   and can be changed as desired.
   To disable rate limiting just delete the <karma/> section.
 -->
       <service id="c2s">
             <load>

<pthsock_client>./pthsock/pthsock_client.so</pthsock_client>
             </load>
             <pthcsock xmlns="jabber:config:pth-csock">
                    <authtime/>
                    <heartbeat/>
                    <karma>
                           <init>10</init>
                           <max>10</max>
                           <inc>1</inc>
                           <dec>1</dec>
                           <penalty>-6</penalty>
                           <restore>10</restore>
                    </karma>
                    <!--
   Use these to listen on particular addresses and/or ports.
   Example: <ip port="5222">127.0.0.1</ip>
   Default is to listen on port 5222 on every interface.
   Remove the <ip/> section to disable non-ssl client connections.
   -->
                    <ip port="5222"/>
                    <!--
   The <ssl/> tag acts pretty much like the <ip/> tag,
   except it defines that SSL is to be used on the
   ports and IP addresses specified. You must specify
   an IP address here, or the connections will fail.
   <ssl port='5223'>127.0.0.1</ssl>
   <ssl port='5224'>192.168.1.100</ssl>
   -->
             </pthcsock>
       </service>
       <!--
 This is the default server error logging component,
 which copies to a file and to STDERR.
 -->
       <log id="elogger">
             <host/>
             <logtype/>
             <format>%d: [%t] (%h): %s</format>
             <file>error.log</file>
             <stderr/>
       </log>
       <!--
 This is the default server record logging component,
 which logs general statistical/tracking data.
 -->
       <log id="rlogger">
```

```
                    <host/>
                    <logtype>record</logtype>
                    <format>%d: [%t] (%h): %s</format>
                    <file>record.log</file>
             </log>
             <!-- The following two services are for handling server-to-server
traffic. -->
             <!-- External asychronous DNS resolver -->
             <service id="dnsrv">
                    <host/>
                    <load>
                           <dnsrv>./dnsrv/dnsrv.so</dnsrv>
                    </load>
                    <dnsrv xmlns="jabber:config:dnsrv">
                           <resend service="_xmpp-server._tcp">s2s</resend>
                           <!-- for supporting XMPP compliant SRV records -->
                           <resend service="_jabber._tcp">s2s</resend>
                           <!-- for supporting old style SRV records -->
                           <resend>s2s</resend>
                    </dnsrv>
             </service>
             <!--
        The following 's2s' config handles server connections and
        dialback hostname verification. The <legacy/> element is
        here to enable communication with old 1.0 servers. The
        karma settings are a little higher here to handle the
        higher traffic of server-to-server connections (read
        the io section below for more details, medium settings).
        -->
             <service id="s2s">
                    <load>
                           <dialback>./dialback/dialback.so</dialback>
                    </load>
                    <dialback xmlns="jabber:config:dialback">
                           <legacy/>
                           <!-- Use these to listen on particular addresses
and/or ports.
          <ip port="7000"/>
          <ip port="5269">127.0.0.1</ip>
          -->
                           <ip port="5269"/>
                           <karma>
                                  <init>50</init>
                                  <max>50</max>
                                  <inc>4</inc>
                                  <dec>1</dec>
                                  <penalty>-5</penalty>
                                  <restore>50</restore>
                           </karma>
                    </dialback>
             </service>
             <!--
        update.jabber.org is long dead but some clients still
        request update information. In order to avoid errors
        in the logs, just drop packages for update.jabber.org.
        -->
             <service id="update.jabber.org">
                    <host>update.jabber.org</host>
                    <null/>
             </service>
             <!--
        If you identified additional agents in the main <service/>
        section (see examples above), you'll need to define each
        of them here using a separate <service/> section for each
        <agent/> you identified. Note that the <agent/> sections
        determine what gets shown to clients that connect to your
        server, whereas the following <service/> sections define
        these services within the server itself. The following are
        examples only, you will need to create/modify them to get
        them working on your Jabber server. See the README files
        for each agent and/or the server howto for further
        information/instructions.
        -->
              <service id="conference.xchat.movesinstitute.org">
                    <load>
                          <conference>./mu-conference-0.6.0/src/mu-
conference.so</conference>
                    </load>
                    <conference xmlns="jabber:config:conference">
                          <public/>
                          <persistent/>
                          <vCard>
                                <FN>Public Chatrooms</FN>
                                <DESC>This service is for public
chatrooms.</DESC>
                                <URL>http://xchat.movesinstitute.org/</URL>
                          </vCard>
                          <history>20</history>
                          <logdir>./logs/</logdir>
                          <sadmin>
                                <user>admin@localhost</user>
                          </sadmin>
                          <notice>
                                <join>has become available</join>
                                <leave>has left</leave>
                                <rename>is now known as</rename>
                          </notice>
                    </conference>
              </service>
              <!-- we're commenting these out, of course :)

        <service id="aim.localhost">
         <accept>
          <ip/>
          <port>7009</port>
          <secret>jabber-rocks</secret>
         </accept>
        </service>

        <service id="yahoo.localhost">
         <accept>
          <ip/>
          <port>9001</port>
          <secret>jabber-rocks</secret>
         </accept>
        </service>

        end of <service/> examples -->
              <!--
        The following <io/> config initializes the top-level
        I/O, otherwise known as MIO (Managed Input/Output).
        -->
              <io>
                    <!-- Set the default karma for *all* sockets -->
                    <!-- definition of terms:
```

```
 * Avg. Throughput - The number of bytes you can
  send every second without incuring any penalty.

 * Burst Allowed - The maximum number of bytes you
  can send in 2 seconds without incurring any penalty.

 * Max Sustained Rate - If you send data as fast as
  you can, you will hit penalty, and will not be
  able to send for 10 seconds; the max sustained
  rate is the average rate you can dump data when
  you are dumping as much data as you can, as fast
  as you can.

 * Seconds to Recover from Burst - The amount of time
  it will take to reach Avg. Throughput capability
  after sending a max burst of data.

 * Penalty Length - The length of your penalty is
  determined according to this formula:
     abs(penalty) * Heartbeat seconds
  E.g., a penalty of -5 and heartbeat of 2 will
  cause your penalty length to be 10 seconds.
  Note that a penalty CANNOT be less than -100,
  otherwise strange things might happen.

-->
           <!-- Example of Low Karma Limits
  Avg. Throughput: 1k-2k/s
  Burst Allowed To: 5.5k/s
  Max Sustained Rate: 485b/s
  Seconds to Recover from Burst: 20
  Penalty Length: 12 seconds
<karma>
 <heartbeat>2</heartbeat>
 <init>10</init>
 <max>10</max>
 <inc>1</inc>
 <dec>1</dec>
 <penalty>-6</penalty>
 <restore>10</restore>
</karma>
-->
           <!-- Example of Medium Karma Limits
  Avg. Throughput: 5k-10k/s
  Burst Allowed: 125.5k/s
  Max Sustained Rate: 12.6k/s
  Seconds to Recover From Burst: 25
  Penalty Length: 10 seconds
<karma>
 <heartbeat>2</heartbeat>
 <init>50</init>
 <max>50</max>
 <inc>4</inc>
 <dec>1</dec>
 <penalty>-5</penalty>
 <restore>50</restore>
</karma>
-->
           <!-- Example of High Karma Limits
  Avg. Throughput: 5k-10k/s
  Burst Allowed: 206k/s
  Max Sustained Rate: 34.3k/s
  Seconds to Recover from Burst: 21
```

```
    Penalty Length: 6 seconds
  <karma>
   <heartbeat>2</heartbeat>
   <init>64</init>
   <max>64</max>
   <inc>6</inc>
   <dec>1</dec>
   <penalty>-3</penalty>
   <restore>64</restore>
  </karma>
  -->
             <!--
  Set rate limits to monitor the number of connection
  attempts from a single IP, any more than [points]
  within [time] will engage the limit. This setting
  applies to all incoming connections to any service,
  unless otherwise overridden by that service.
  -->
             <rate points="5" time="25"/>
             <!--
  The following section initializes SSL for top-level I/O.
  This works only when the server is compiled with openssl!
  Use IPs here or connections will fail.
  -->
             <!--
  <ssl>
   <key ip='192.168.1.1'>/path/to/cert_and_key.pem</key>
   <key ip='192.168.1.100'>/path/to/other/cert_and_key.pem</key>
  </ssl>
  -->
             <!--
  The following section is used to allow or deny
  communications from specified IP networks or
  addressses. If there is no <allow/> section,
  then *all* IPs will be allowed to connect. If
  you allow one block, then only that block may
  connect. Note that <allow/> is checked before
  <deny/>, so if a specific address is allowed
  but the network for that address is denied,
  then that address will still be denied.
  -->
             <!--
  <allow><ip>127.0.0.0</ip><mask>255.255.255.0</mask></allow>
  <allow><ip>12.34.56.78</ip></allow>
  <deny><ip>22.11.44.0</ip><mask>255.255.255.0</mask></deny>
  -->
       </io>
       <!--
 This specifies the file to store the pid of the process in.
 -->
       <pidfile>./jabber.pid</pidfile>
</jabber>
```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX C. XTC EXAMPLE TACTICAL MESSAGE SCHEMA

The following XML schema was written by Brian Hittner to validate example tactical messages for XTC.

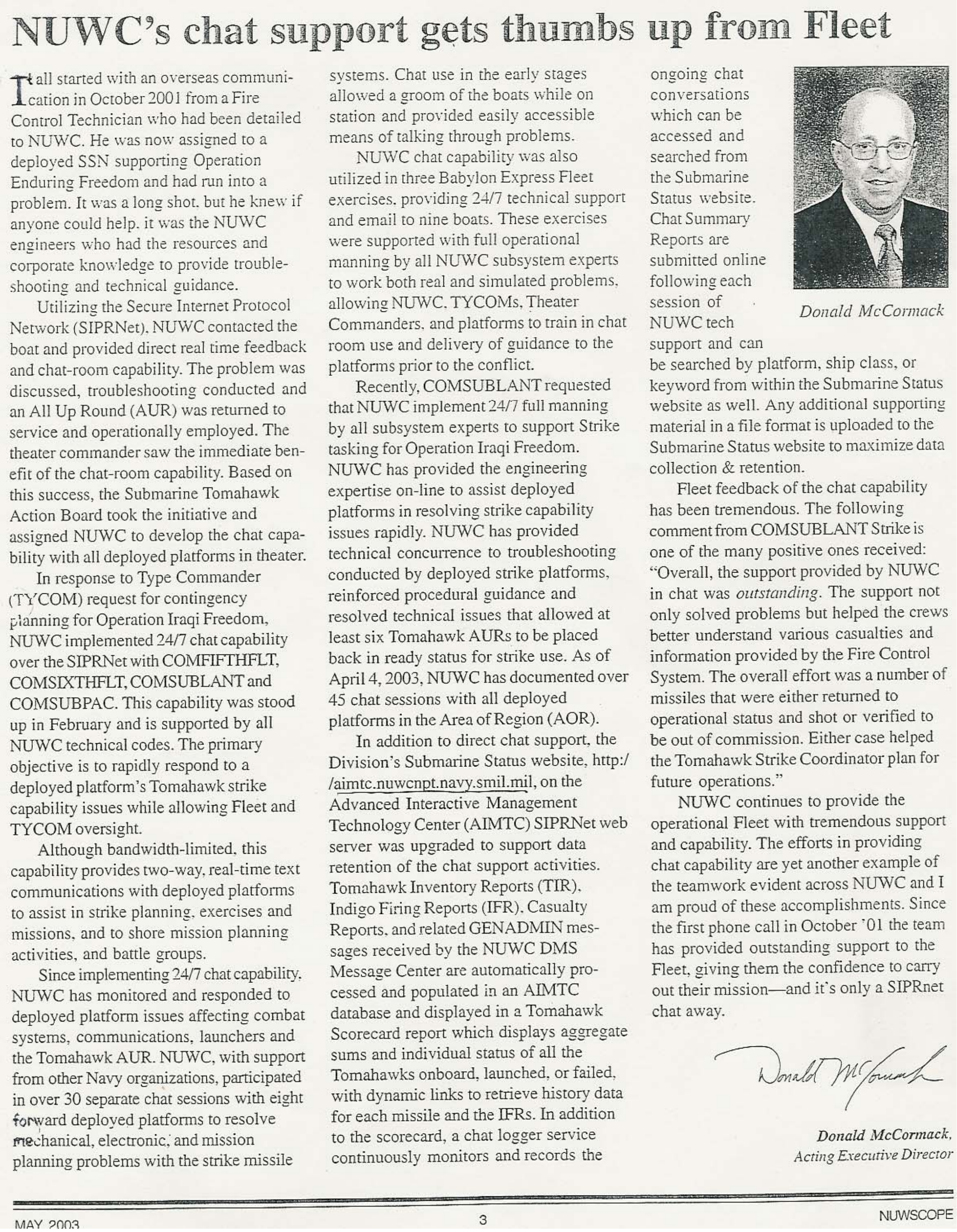
```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema
targetNamespace="http://www.movesinstitute.org/xtc/schemas/TacticalMessageTempl
ates1.0.xsd"
xmlns:xtc="http://www.movesinstitute.org/xtc/schemas/TacticalMessageTemplates1.
0.xsd" xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified">
   <!-- Blue1 Report Elements (aka Spot Report or Contact Report) -->
   <xs:element name="Observer">
      <xs:complexType mixed="true">
         <xs:attribute name="input" fixed="true"/>
         <xs:attribute name="label" fixed="Observer"/>
         <xs:attribute name="required" fixed="true"/>
      </xs:complexType>
   </xs:element>
   <xs:element name="Size">
      <xs:complexType mixed="true">
         <xs:attribute name="input" fixed="true"/>
         <xs:attribute name="label" fixed="Size"/>
         <xs:attribute name="required" fixed="true"/>
      </xs:complexType>
   </xs:element>
   <xs:element name="Activity">
      <xs:complexType mixed="true">
         <xs:attribute name="input" fixed="true"/>
         <xs:attribute name="label" fixed="Activity"/>
         <xs:attribute name="required" fixed="true"/>
      </xs:complexType>
   </xs:element>
   <xs:element name="Location">
      <xs:complexType mixed="true">
         <xs:attribute name="input" fixed="true"/>
         <xs:attribute name="label" fixed="Location"/>
         <xs:attribute name="required" fixed="true"/>
      </xs:complexType>
   </xs:element>
   <xs:element name="Unit">
      <xs:complexType mixed="true">
         <xs:attribute name="input" fixed="true"/>
         <xs:attribute name="label" fixed="Unit"/>
         <xs:attribute name="required" fixed="false"/>
      </xs:complexType>
   </xs:element>
   <xs:element name="DateTime">
      <xs:complexType mixed="true">
         <xs:attribute name="input" fixed="true"/>
         <xs:attribute name="label" fixed="Time"/>
         <xs:attribute name="required" fixed="true"/>
      </xs:complexType>
   </xs:element>
   <xs:element name="Equipment">
      <xs:complexType mixed="true">
         <xs:attribute name="input" fixed="true"/>
         <xs:attribute name="label" fixed="Equipment"/>
         <xs:attribute name="required" fixed="false"/>
      </xs:complexType>
   </xs:element>
   <xs:element name="Description">
      <xs:complexType>
         <xs:sequence>
            <xs:element ref="xtc:Size"/>
            <xs:element ref="xtc:Activity"/>
            <xs:element ref="xtc:Location"/>
            <xs:element ref="xtc:Unit"/>
            <xs:element ref="xtc:DateTime"/>
            <xs:element ref="xtc:Equipment"/>
         </xs:sequence>
      </xs:complexType>
   </xs:element>
   <xs:element name="Action">
      <xs:complexType mixed="true">
         <xs:attribute name="input" fixed="true"/>
         <xs:attribute name="label" fixed="Observer Action"/>
         <xs:attribute name="required" fixed="true"/>
      </xs:complexType>
   </xs:element>
   <xs:element name="SpotReport">
      <xs:complexType>
         <xs:sequence>
            <xs:element ref="xtc:head"/>
            <xs:element name="body">
               <xs:complexType>
                  <xs:sequence>
                     <xs:element ref="xtc:Observer"/>
                     <xs:element ref="xtc:Description"/>
                     <xs:element ref="xtc:Action"/>
                  </xs:sequence>
               </xs:complexType>
            </xs:element>
         </xs:sequence>
      </xs:complexType>
   </xs:element>
   <!-- End Blue 1 Elements -->
   <!-- Call For Fire Elements -->
   <!-- There would be an Observer declaration here, but it was already declared
above -->
   <xs:element name="TargetLocation">
      <xs:complexType mixed="true">
         <xs:attribute name="input" fixed="true"/>
         <xs:attribute name="label" fixed="Target Location"/>
         <xs:attribute name="required" fixed="true"/>
      </xs:complexType>
   </xs:element>
   <xs:element name="TargetDescription">
      <xs:complexType mixed="true">
         <xs:attribute name="input" fixed="true"/>
         <xs:attribute name="label" fixed="Target Description"/>
         <xs:attribute name="required" fixed="true"/>
      </xs:complexType>
   </xs:element>
   <xs:element name="EngagementMethod">
      <xs:complexType mixed="true">
         <xs:attribute name="input" fixed="true"/>
         <xs:attribute name="label" fixed="Engagement Method"/>
         <xs:attribute name="required" fixed="true"/>
      </xs:complexType>
   </xs:element>
   <xs:element name="MethodOfFire">
      <xs:complexType mixed="true">
         <xs:attribute name="input" fixed="true"/>
         <xs:attribute name="label" fixed="Method of Fire"/>
         <xs:attribute name="required" fixed="true"/>
      </xs:complexType>
   </xs:element>
   <xs:element name="CallForFire">
      <xs:complexType>
         <xs:sequence>
            <xs:element ref="xtc:head"/>
            <xs:element name="body">
               <xs:complexType>
                  <xs:sequence>
                     <xs:element ref="xtc:Observer"/>
                     <xs:element ref="xtc:TargetLocation"/>
                     <xs:element ref="xtc:TargetDescription"/>
                     <xs:element ref="xtc:EngagementMethod"/>
                     <xs:element ref="xtc:MethodOfFire"/>
                  </xs:sequence>
               </xs:complexType>
            </xs:element>
         </xs:sequence>
      </xs:complexType>
   </xs:element>
   <!-- End Call For Fire Elements -->
   <!-- Blue 2a Report Elements (Ammunition Request) -->
   <!-- There would be a Unit declaration here, but it was already declared
above -->
   <!-- There would be a Date Time declaration here, but it was already declared
above -->
   <xs:element name="HEAT105mm">
      <xs:complexType mixed="true">
         <xs:attribute name="input" fixed="true"/>
         <xs:attribute name="label" fixed="105mm HEAT"/>
         <xs:attribute name="required" fixed="false"/>
      </xs:complexType>
   </xs:element>
   <xs:element name="APDS105mm">
      <xs:complexType mixed="true">
         <xs:attribute name="input" fixed="true"/>
         <xs:attribute name="label" fixed="105mm APDS"/>
         <xs:attribute name="required" fixed="false"/>
      </xs:complexType>
   </xs:element>
   <xs:element name="HEAT120mm">
      <xs:complexType mixed="true">
         <xs:attribute name="input" fixed="true"/>
         <xs:attribute name="label" fixed="120mm HEAT"/>
         <xs:attribute name="required" fixed="false"/>
      </xs:complexType>
   </xs:element>
   <xs:element name="APDS120mm">
      <xs:complexType mixed="true">
         <xs:attribute name="input" fixed="true"/>
         <xs:attribute name="label" fixed="120mm APDS"/>
         <xs:attribute name="required" fixed="false"/>
      </xs:complexType>
   </xs:element>
   <xs:element name="HEDP40mmM203Box">
      <xs:complexType mixed="true">
         <xs:attribute name="input" fixed="true"/>
         <xs:attribute name="label" fixed="40mm M203 HEDP, Box"/>
         <xs:attribute name="required" fixed="false"/>
      </xs:complexType>
   </xs:element>
   <xs:element name="Cal50M85Box">
      <xs:complexType mixed="true">
         <xs:attribute name="input" fixed="true"/>
         <xs:attribute name="label" fixed=".50 Cal M85, Box"/>
         <xs:attribute name="required" fixed="false"/>
      </xs:complexType>
   </xs:element>
   <xs:element name="Cal50M2Box100">
      <xs:complexType mixed="true">
         <xs:attribute name="input" fixed="true"/>
         <xs:attribute name="label" fixed=".50 Cal M2, Box of 100"/>
         <xs:attribute name="required" fixed="false"/>
      </xs:complexType>
   </xs:element>
   <xs:element name="CoaxM607.62mmBox400">
      <xs:complexType mixed="true">
         <xs:attribute name="input" fixed="true"/>
         <xs:attribute name="label" fixed="7.62mm M607 Coax, Box of 400"/>
         <xs:attribute name="required" fixed="false"/>
      </xs:complexType>
   </xs:element>
   <xs:element name="HighExplosive4.2in">
      <xs:complexType mixed="true">
         <xs:attribute name="input" fixed="true"/>
         <xs:attribute name="label" fixed="4.2 in. High Explosive Mortar"/>
         <xs:attribute name="required" fixed="false"/>
      </xs:complexType>
   </xs:element>
   <xs:element name="WhitePhosphorous4.2in">
      <xs:complexType mixed="true">
         <xs:attribute name="input" fixed="true"/>
         <xs:attribute name="label" fixed="4.2 in. White Phosphorous Mortar"/>
         <xs:attribute name="required" fixed="false"/>
      </xs:complexType>
   </xs:element>
   <xs:element name="Illumination4.2in">
      <xs:complexType mixed="true">
         <xs:attribute name="input" fixed="true"/>
         <xs:attribute name="label" fixed="4.2 in. Illumination Mortar"/>
         <xs:attribute name="required" fixed="false"/>
      </xs:complexType>
   </xs:element>
   <xs:element name="ProximityFuze4.2in">
      <xs:complexType mixed="true">
         <xs:attribute name="input" fixed="true"/>
         <xs:attribute name="label" fixed="4.2 in. Mortar Proximity Fuze"/>
         <xs:attribute name="required" fixed="false"/>
      </xs:complexType>
   </xs:element>
   <xs:element name="PointDetonationFuze4.2in">
      <xs:complexType mixed="true">
         <xs:attribute name="input" fixed="true"/>
         <xs:attribute name="label" fixed="4.2 in. Mortar Point Detonation
Fuze"/>
         <xs:attribute name="required" fixed="false"/>
      </xs:complexType>
   </xs:element>
   <xs:element name="ProximityFuze81mm">
      <xs:complexType mixed="true">
         <xs:attribute name="input" fixed="true"/>
      <xs:attribute name="label" fixed="81mm Mortar Proximity Fuze"/>
      <xs:attribute name="required" fixed="false"/>
   </xs:complexType>
</xs:element>
<xs:element name="PointDetonationFuze81mm">
   <xs:complexType mixed="true">
      <xs:attribute name="input" fixed="true"/>
      <xs:attribute name="label" fixed="81mm Mortar Point Detonation Fuze"/>
      <xs:attribute name="required" fixed="false"/>
   </xs:complexType>
</xs:element>
<xs:element name="BlastingCapNonElectricBox">
   <xs:complexType mixed="true">
      <xs:attribute name="input" fixed="true"/>
      <xs:attribute name="label" fixed="Blasting Cap, Non-Electric, Box"/>
      <xs:attribute name="required" fixed="false"/>
   </xs:complexType>
</xs:element>
<xs:element name="FuzeIgnitor">
   <xs:complexType mixed="true">
      <xs:attribute name="input" fixed="true"/>
      <xs:attribute name="label" fixed="Fuze Ignitor"/>
      <xs:attribute name="required" fixed="false"/>
   </xs:complexType>
</xs:element>
<xs:element name="Ball5.56mmBox400">
   <xs:complexType mixed="true">
      <xs:attribute name="input" fixed="true"/>
      <xs:attribute name="label" fixed="5.56mm Ball, Box of 400"/>
      <xs:attribute name="required" fixed="false"/>
   </xs:complexType>
</xs:element>
<xs:element name="Tracer5.56mmBox400">
   <xs:complexType mixed="true">
      <xs:attribute name="input" fixed="true"/>
      <xs:attribute name="label" fixed="5.56mm Tracer, Box of 400"/>
      <xs:attribute name="required" fixed="false"/>
   </xs:complexType>
</xs:element>
<xs:element name="FragmentationGrenade">
   <xs:complexType mixed="true">
      <xs:attribute name="input" fixed="true"/>
      <xs:attribute name="label" fixed="Grenade, Fragmentation"/>
      <xs:attribute name="required" fixed="false"/>
   </xs:complexType>
</xs:element>
<xs:element name="SmokeGrenade">
   <xs:complexType mixed="true">
      <xs:attribute name="input" fixed="true"/>
      <xs:attribute name="label" fixed="Grenade, Smoke, High Concentration"/>
      <xs:attribute name="required" fixed="false"/>
   </xs:complexType>
</xs:element>
<xs:element name="ThermiteGrenade">
   <xs:complexType mixed="true">
      <xs:attribute name="input" fixed="true"/>
      <xs:attribute name="label" fixed="Grenade, Thermite"/>
      <xs:attribute name="required" fixed="false"/>
   </xs:complexType>
</xs:element>
<xs:element name="Grenade40mmHighExplosiveBox">
   <xs:complexType mixed="true">
      <xs:attribute name="input" fixed="true"/>
```

```
         <xs:attribute name="label" fixed="40mm Grenade, High Explosive, Box"/>
         <xs:attribute name="required" fixed="false"/>
      </xs:complexType>
   </xs:element>
   <xs:element name="Grenade40mmWhitePhosphorousBox">
      <xs:complexType mixed="true">
         <xs:attribute name="input" fixed="true"/>
         <xs:attribute name="label" fixed="40mm Grenade, White Phosphorous,
Box"/>
         <xs:attribute name="required" fixed="false"/>
      </xs:complexType>
   </xs:element>
   <xs:element name="Grenade40mmArmorPiercingBox">
      <xs:complexType mixed="true">
         <xs:attribute name="input" fixed="true"/>
         <xs:attribute name="label" fixed="40mm Grenade, Armor Piercing, Box"/>
         <xs:attribute name="required" fixed="false"/>
      </xs:complexType>
   </xs:element>
   <xs:element name="Ball9mmBox50">
      <xs:complexType mixed="true">
         <xs:attribute name="input" fixed="true"/>
         <xs:attribute name="label" fixed="9mm Ball, Box of 50"/>
         <xs:attribute name="required" fixed="false"/>
      </xs:complexType>
   </xs:element>
   <xs:element name="LightAntitankWeaponM72">
      <xs:complexType mixed="true">
         <xs:attribute name="input" fixed="true"/>
         <xs:attribute name="label" fixed="LAW, Light Antitank Weapon, M72"/>
         <xs:attribute name="required" fixed="false"/>
      </xs:complexType>
   </xs:element>
   <xs:element name="Dragon">
      <xs:complexType mixed="true">
         <xs:attribute name="input" fixed="true"/>
         <xs:attribute name="label" fixed="Missile, Dragon"/>
         <xs:attribute name="required" fixed="false"/>
      </xs:complexType>
   </xs:element>
   <xs:element name="TOWIIB">
      <xs:complexType mixed="true">
         <xs:attribute name="input" fixed="true"/>
         <xs:attribute name="label" fixed="TOW II B Missile"/>
         <xs:attribute name="required" fixed="false"/>
      </xs:complexType>
   </xs:element>
   <xs:element name="StingerMissile">
      <xs:complexType mixed="true">
         <xs:attribute name="input" fixed="true"/>
         <xs:attribute name="label" fixed="Missile, Stinger"/>
         <xs:attribute name="required" fixed="false"/>
      </xs:complexType>
   </xs:element>
   <xs:element name="AntiTankMine">
      <xs:complexType mixed="true">
         <xs:attribute name="input" fixed="true"/>
         <xs:attribute name="label" fixed="Mine, Anti-Tank"/>
         <xs:attribute name="required" fixed="false"/>
      </xs:complexType>
   </xs:element>
   <xs:element name="AntiPersonnelMine">
      <xs:complexType mixed="true">
```

```
         <xs:attribute name="input" fixed="true"/>
         <xs:attribute name="label" fixed="Mine, Anti-Personnel"/>
         <xs:attribute name="required" fixed="false"/>
      </xs:complexType>
   </xs:element>
   <xs:element name="ClaymoreMine">
      <xs:complexType mixed="true">
         <xs:attribute name="input" fixed="true"/>
         <xs:attribute name="label" fixed="Mine, Claymore"/>
         <xs:attribute name="required" fixed="false"/>
      </xs:complexType>
   </xs:element>
   <xs:element name="HighExplosive25mmBox50">
      <xs:complexType mixed="true">
         <xs:attribute name="input" fixed="true"/>
         <xs:attribute name="label" fixed="25mm, High Explosive, Box of 50"/>
         <xs:attribute name="required" fixed="false"/>
      </xs:complexType>
   </xs:element>
   <xs:element name="ArmorPiercing25mmBox50">
      <xs:complexType mixed="true">
         <xs:attribute name="input" fixed="true"/>
         <xs:attribute name="label" fixed="25mm, Armor Piercing, Box of 50"/>
         <xs:attribute name="required" fixed="false"/>
      </xs:complexType>
   </xs:element>
   <xs:element name="HighExplosivePlastic165mm">
      <xs:complexType mixed="true">
         <xs:attribute name="input" fixed="true"/>
         <xs:attribute name="label" fixed="165mm, High Explosive Plastic"/>
         <xs:attribute name="required" fixed="false"/>
      </xs:complexType>
   </xs:element>
   <xs:element name="AmmunitionRequest">
      <xs:complexType>
         <xs:sequence>
            <xs:element ref="xtc:head"/>
            <xs:element name="body">
               <xs:complexType>
                  <xs:sequence>
                     <xs:element ref="xtc:Unit"/>
                     <xs:element ref="xtc:DateTime"/>
                     <xs:element ref="xtc:HEAT105mm" minOccurs="0"/>
                     <xs:element ref="xtc:APDS105mm" minOccurs="0"/>
                     <xs:element ref="xtc:HEAT120mm" minOccurs="0"/>
                     <xs:element ref="xtc:APDS120mm" minOccurs="0"/>
                     <xs:element ref="xtc:HEDP40mmM203Box" minOccurs="0"/>
                     <xs:element ref="xtc:Cal50M85Box" minOccurs="0"/>
                     <xs:element ref="xtc:Cal50M2Box100" minOccurs="0"/>
                     <xs:element ref="xtc:CoaxM607.62mmBox400" minOccurs="0"/>
                     <xs:element ref="xtc:HighExplosive4.2in" minOccurs="0"/>
                     <xs:element ref="xtc:WhitePhosphorous4.2in" minOccurs="0"/>
                     <xs:element ref="xtc:Illumination4.2in" minOccurs="0"/>
                     <xs:element ref="xtc:ProximityFuze4.2in" minOccurs="0"/>
                     <xs:element ref="xtc:PointDetonationFuze4.2in"
minOccurs="0"/>
                     <xs:element ref="xtc:ProximityFuze81mm" minOccurs="0"/>
                     <xs:element ref="xtc:PointDetonationFuze81mm"
minOccurs="0"/>
                     <xs:element ref="xtc:BlastingCapNonElectricBox"
minOccurs="0"/>
                     <xs:element ref="xtc:FuzeIgnitor" minOccurs="0"/>
                     <xs:element ref="xtc:Ball5.56mmBox400" minOccurs="0"/>
                     <xs:element ref="xtc:Tracer5.56mmBox400" minOccurs="0"/>
                     <xs:element ref="xtc:FragmentationGrenade" minOccurs="0"/>
                     <xs:element ref="xtc:SmokeGrenade" minOccurs="0"/>
                     <xs:element ref="xtc:ThermiteGrenade" minOccurs="0"/>
                     <xs:element ref="xtc:Grenade40mmHighExplosiveBox"
minOccurs="0"/>
                     <xs:element ref="xtc:Grenade40mmWhitePhosphorousBox"
minOccurs="0"/>
                     <xs:element ref="xtc:Grenade40mmArmorPiercingBox"
minOccurs="0"/>
                     <xs:element ref="xtc:Ball9mmBox50" minOccurs="0"/>
                     <xs:element ref="xtc:LightAntitankWeaponM72" minOccurs="0"/>
                     <xs:element ref="xtc:Dragon" minOccurs="0"/>
                     <xs:element ref="xtc:TOWIIB" minOccurs="0"/>
                     <xs:element ref="xtc:StingerMissile" minOccurs="0"/>
                     <xs:element ref="xtc:AntiTankMine" minOccurs="0"/>
                     <xs:element ref="xtc:AntiPersonnelMine" minOccurs="0"/>
                     <xs:element ref="xtc:ClaymoreMine" minOccurs="0"/>
                     <xs:element ref="xtc:HighExplosive25mmBox50" minOccurs="0"/>
                     <xs:element ref="xtc:ArmorPiercing25mmBox50" minOccurs="0"/>
                     <xs:element ref="xtc:HighExplosivePlastic165mm"
minOccurs="0"/>
                  </xs:sequence>
               </xs:complexType>
            </xs:element>
         </xs:sequence>
      </xs:complexType>
   </xs:element>
   <!-- End Blue 2a Report Elements (Ammunition Request) -->
   <!-- Tactical Message Elements -->
   <xs:element name="head">
      <xs:complexType>
         <xs:attribute name="messageType" type="xs:string"/>
         <xs:attribute name="name" type="xs:string"/>
         <xs:attribute name="country" type="xs:string"/>
         <xs:attribute name="branch" type="xs:string"/>
      </xs:complexType>
   </xs:element>
   <xs:element name="TacticalMessage">
      <xs:complexType>
         <xs:choice>
            <xs:element ref="xtc:SpotReport"/>
            <xs:element ref="xtc:CallForFire"/>
            <xs:element ref="xtc:AmmunitionRequest"/>
         </xs:choice>
      </xs:complexType>
   </xs:element>
   <!-- End Tactical Message Elements -->
   <xs:element name="xtc">
      <xs:complexType>
         <xs:choice maxOccurs="unbounded">
            <xs:element ref="xtc:TacticalMessage"/>
         </xs:choice>
      </xs:complexType>
   </xs:element>
</xs:schema>
```

APPENDIX D. NUWC'S CHAT SUPPORT GETS THUMBS UP FROM FLEET

McCormack, D., *NUWSCOPE*, laboratory newsletter, May 2003, page 3.



NUWC's chat support gets thumbs up from Fleet

It all started with an overseas communication in October 2001 from a Fire Control Technician who had been detailed to NUWC. He was now assigned to a deployed SSN supporting Operation Enduring Freedom and had run into a problem. It was a long shot. but he knew if anyone could help. it was the NUWC engineers who had the resources and corporate knowledge to provide troubleshooting and technical guidance.

Utilizing the Secure Internet Protocol Network (SIPRNet). NUWC contacted the boat and provided direct real time feedback and chat-room capability. The problem was discussed, troubleshooting conducted and an All Up Round (AUR) was returned to service and operationally employed. The theater commander saw the immediate benefit of the chat-room capability. Based on this success, the Submarine Tomahawk Action Board took the initiative and assigned NUWC to develop the chat capability with all deployed platforms in theater.

In response to Type Commander (TYCOM) request for contingency planning for Operation Iraqi Freedom, NUWC implemented 24/7 chat capability over the SIPRNet with COMFIFTHFLT, COMSIXTHFLT, COMSUBLANT and COMSUBPAC. This capability was stood up in February and is supported by all NUWC technical codes. The primary objective is to rapidly respond to a deployed platform's Tomahawk strike capability issues while allowing Fleet and TYCOM oversight.

Although bandwidth-limited, this capability provides two-way, real-time text communications with deployed platforms to assist in strike planning, exercises and missions, and to shore mission planning activities, and battle groups.

Since implementing 24/7 chat capability. NUWC has monitored and responded to deployed platform issues affecting combat systems, communications, launchers and the Tomahawk AUR. NUWC, with support from other Navy organizations, participated in over 30 separate chat sessions with eight forward deployed platforms to resolve mechanical, electronic, and mission planning problems with the strike missile systems. Chat use in the early stages allowed a groom of the boats while on station and provided easily accessible means of talking through problems.

NUWC chat capability was also utilized in three Babylon Express Fleet exercises. providing 24/7 technical support and email to nine boats. These exercises were supported with full operational manning by all NUWC subsystem experts to work both real and simulated problems, allowing NUWC. TYCOMs, Theater Commanders, and platforms to train in chat room use and delivery of guidance to the platforms prior to the conflict.

Recently, COMSUBLANT requested that NUWC implement 24/7 full manning by all subsystem experts to support Strike tasking for Operation Iraqi Freedom. NUWC has provided the engineering expertise on-line to assist deployed platforms in resolving strike capability issues rapidly. NUWC has provided technical concurrence to troubleshooting conducted by deployed strike platforms, reinforced procedural guidance and resolved technical issues that allowed at least six Tomahawk AURs to be placed back in ready status for strike use. As of April 4, 2003, NUWC has documented over 45 chat sessions with all deployed platforms in the Area of Region (AOR).

In addition to direct chat support, the Division's Submarine Status website, http://aimtc.nuwcnpt.navy.smil.mil, on the Advanced Interactive Management Technology Center (AIMTC) SIPRNet web server was upgraded to support data retention of the chat support activities. Tomahawk Inventory Reports (TIR), Indigo Firing Reports (IFR). Casualty Reports, and related GENADMIN messages received by the NUWC DMS Message Center are automatically processed and populated in an AIMTC database and displayed in a Tomahawk Scorecard report which displays aggregate sums and individual status of all the Tomahawks onboard, launched, or failed, with dynamic links to retrieve history data for each missile and the IFRs. In addition to the scorecard, a chat logger service continuously monitors and records the ongoing chat conversations which can be accessed and searched from the Submarine Status website. Chat Summary Reports are submitted online following each session of NUWC tech support and can be searched by platform, ship class, or keyword from within the Submarine Status website as well. Any additional supporting material in a file format is uploaded to the Submarine Status website to maximize data collection & retention.

Donald McCormack

Fleet feedback of the chat capability has been tremendous. The following comment from COMSUBLANT Strike is one of the many positive ones received: "Overall, the support provided by NUWC in chat was *outstanding*. The support not only solved problems but helped the crews better understand various casualties and information provided by the Fire Control System. The overall effort was a number of missiles that were either returned to operational status and shot or verified to be out of commission. Either case helped the Tomahawk Strike Coordinator plan for future operations."

NUWC continues to provide the operational Fleet with tremendous support and capability. The efforts in providing chat capability are yet another example of the teamwork evident across NUWC and I am proud of these accomplishments. Since the first phone call in October '01 the team has provided outstanding support to the Fleet, giving them the confidence to carry out their mission—and it's only a SIPRnet chat away.

Donald McCormack

Donald McCormack,
Acting Executive Director

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX E. NAVY-MARINE CORPS UNCLASSIFIED TRUSTED NETWORK PROTECTION POLICY

1. The Navy-Marine Corps Unclassified Trusted Network Protection (UTNProtect) Policy[1] is the guidance for interconnection between Navy and Marine Corps trusted networks (complies with this policy) and untrusted networks (NIPRNET). The basic policy is one of "deny by exception" for firewall access.

2. Section 4 of the UTNProtect Policy was downgraded to unclassified.[2] This section lists the current ports that are open to inbound/outbound internet traffic to untrusted networks.

3. Traffic between trusted networks is allowed. A request at each end should be made to the respective firewall administrator to open a specified port for use.

4. Currently, at NPS, port 443 (SSL) is open for use. However, the firewall administrator should be contacted prior to use to ensure adequate capabilities are available.

5. Requests for specific ports to be opened with accompanying justification are to be submitted to NCO (N6142) and SPAWAR PMW161 via message traffic. Further guidance is found in UNTProtect Policy.[3]

[1] CNO N614/HQMC C4, dtd 31Oct2002

[2] CNO ltr 5239 Ser N614/3U605717 dtd 30May2003

[3] https://infosec.navy.mil/pub/docs/documents/navyn/utn/Navy_Marine_Corps _UTNProtect_Policy_(FOUO)_31_Oct02.doc

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX F. TASK FORCE WEB SECURITY REQUIREMENTS

The following security requirements come from Task Force Webs' *Application Development Guide, Appendix G: Application Security*, Version 1.12, February 10, 2003.

There are two important interfaces that must be secured when migrating applications and content into the Navy Enterprise Portal (NEP): the portlet interface and the user-facing service (UFS) interface to the portal. It is the responsibility of the NEP to secure the portlet interface. It is the responsibility of the application owner to secure the UFS interface, with the assistance of the services provided by the NEP. Furthermore, the application owner must address application-specific security measures for the UFS (and the hardware supporting it) to meet the requirements of the application's developmental DAA.

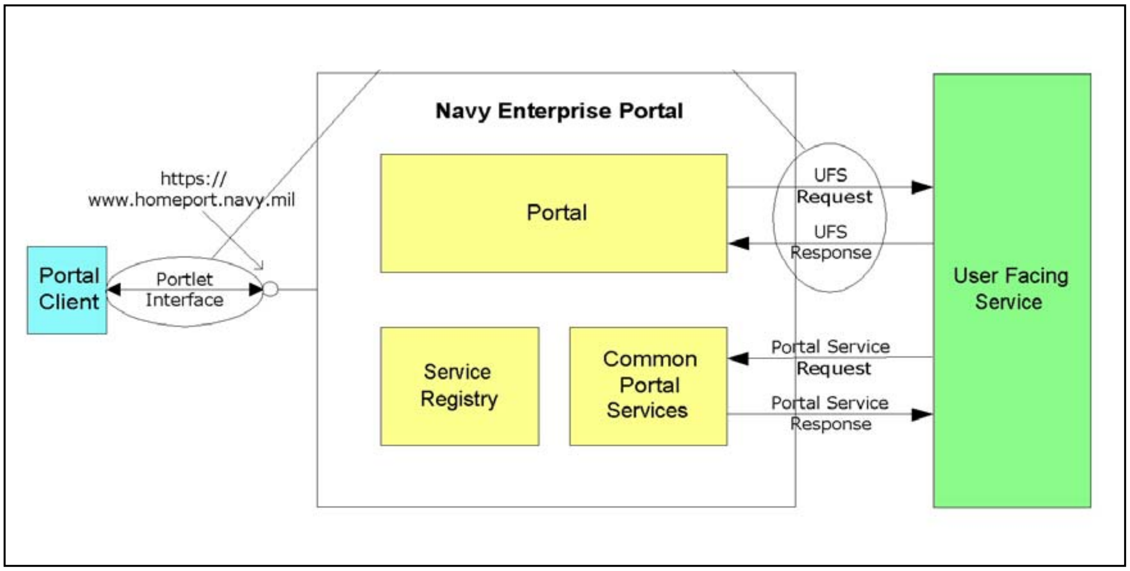


Figure 80. Scope of application security for Task Force Web. Adapted from [TFWeb 03].

The most significant decision made by the application owner is whether the security used by the UFS interface to the portal provides an adequate level of trust for the data accessed through the UFS. If the owner chooses to not trust these mechanisms, he may choose to require a separate authentication to the application. If the application reauthenticates portal users, it must do so in a portal-friendly manner (such as not automatically popping up new windows).

1. Portlet Interface Security Description – Current Architecture

Note that the NEP does not support all certificates issued by DoD certificate authorities (CAs). Currently, only certificates in the hierarchy based on the DoD Class 3 Root CA certificate are supported (this includes DoD Class 3 CA-3, DoD Class 3 CA-4, and DoD Class 3 CAC CA.) Certificates in the hierarchy based on the DoD PKI Med Root CA are not supported (this includes Med CA-1 and Med CA-2.)

The portal performs a two-factor authentication:

The portal client is asked to present a login and password.

- Encrypted connection – the portal can interface with the application using an encrypted connection. Currently, the NEP can only support SSL connections over the UFS interface. Other encryption standards such as TLS are not supported.
- The certificates are checked against the appropriate DoD root trust chain and certificate revocation list to ensure validity.
- The user enters his or her common identity UserID and password.
- The UserID is authenticated against the Naval Global Directory Service (NGDS) and user roles are passed to the portal. User roles determine which library of services is made available to the user.

After authentication, the portal can then provide the common identity UserID to any portal service via HTTP or HTTPS by the portal request interface (PRI) Request or by authorization HTTP header field.

2. Portlet Interface Security Description - Objective Architecture

A COTS single-sign-on (SSO) product is the next step in accessing security implementation for services integrated into the NEP. Presently, a product has not been implemented for use. Guidance for the implementation of the SSO product will be provided in the future. Contact the AMCS for further information.

3. User-Facing Services (UFS) Interface Security Implementation

The NEP can provide two items for applications and services via the UFS interface:

- Encrypted connection. The portal can interface to the application using an encrypted connection. Currently, NEP policy states that the portal shall only support SSL connections to UFS or DOS.
- Common identity. The portal sends the common identity through the PRI request and/or authorization header field (recommended) in the HTTP header.

The multiple combinations of the above items are listed in Figure 81.

Common Identity Sent	Encrypted Connection (SSL)
No	No
Yes	No
No	Yes
Yes	Yes

Figure 81. Security implementation combinations for user-facing services. [TFWeb 03]

The following examples use different combinations of portal capabilities to secure the UFS interface. These examples are not all-inclusive. Each has advantages and disadvantages and different trust levels. An example using the future SSO product is also included. The decision on which combination to use is based on information needed within the application/service. Section V of Web Site Administration Policies & Procedures, contains examples and best practices for Web information security. [Policies 98]

4. Example 1: A General Public Service

The portal provides no common-identity information or PRI request header, nor is the UFS interface encrypted. Examples: Early Bird News Service, National Weather Service information.

Advantages	Disadvantages
Higher Performance	Higher risks, since host server is exposed to the general Internet population; however equal to current risk assumed.
No application userID/password infrastructure required	Should not be used for applications requiring any form of authentication/authorization.
Easy to implement	Requires periodic synchronization

Figure 82. Advantages and Disadvantages of using a General Public Service to secure a UFS interface. [TFWeb 03]

Users of a service using this security methodology are considered to be anonymous, because they cannot be authenticated over the unencrypted UFS interface. Data passed over the UFS interface will also be in the clear. This methodology provides no trust.

5. Example 2: An SSL Service

The portal provides no common identity information or PRI request header, but the UFS interface is encrypted.

Note: Applications and services that are designed to be inaccessible by the general public must obtain a DoD server certificate and use two-way SSL per CNO message DTG 301704Z NOV 00. Example: https://www.infosec.navy.mil

Advantages	Disadvantages
Most browsers support SSL.	Requires a DoD PKI server certificate.
Existing means of application authentication can still be used.	Lower performance.

Figure 83. Advantages and Disadvantages of using a SSL to secure a UFS interface. [TFWeb 03]

To provide assurance of the user's identity, the service must re-authenticate the user independently of the portal, in a portal-friendly manner. Data sent over the UFS interface will also be encrypted, assuring that it will not be compromised in transit. With user re-authentication, this methodology provides a high level of trust. Without authentication, this methodology provides no trust of user identity, but a high level of trust for data.

6. Example 3: Portal-supplied Common Identity without SSL

The portal provides common identity information; however, the UFS interface is not encrypted.

Note that the application/service can use the common identity as a means of identifying users and possibly tailoring its functionality based on common identity. For example, an application or service receives a common identity for tracking information such as date of last login. Implementing the common identity on existing applications requires a mapping from the common identity to existing application userID. Authentication and authorization of application rights without SSL is prohibited by current policy.

Advantages	Disadvantages:
Common identity is available to provide personalization of the service.	The existing applications will need modifications to support the common identity (Mapping).
Higher performance without SSL.	Must read header field or parse PRI to determine common identity.
	Should not be used for applications requiring any form of authentication/authorization.

Figure 84. Advantages and disadvantages of using a portal-supplied common identity without SSL to secure a UFS interface. [TFWeb 03]

Users of a service employing this security methodology are considered to be anonymous, because they cannot be authenticated over the unencrypted UFS interface. However, if the application owner and the DAA accept the risk, the common identity could be used for personalization. Data passed over the UFS interface will also be in the clear. This methodology provides a very low level of trust.

7. Example 4: Portal-supplied Common Identity with SSL

The portal provides common identity information and the UFS interface is encrypted. The application/service uses the common identity as a means of identifying users, tailoring its functionality, and possibly assigning application-local roles to those users. A use of this combination would be to mimic an SSO capability. An application/service may choose to accept the passed common identity to allow access and perform authorization for that user.

Advantages	Disadvantages
Support for the user's identity is now shifted away from the application/service developer. The application owner no longer needs to manage user passwords, but must still manage users for means of authorization.	The existing applications will need modifications to support the common identity. Application local user information will need to be stored in a local database.
Common identity may be used for re-authentication to the application/service, because passwords are sent encrypted.	Requires a DoD server certificate.
Can eliminate multiple login screens.	Lower performance.

Figure 85. Advantages and disadvantages of using a portal-supplied common identity with SSL to secure a UFS interface. [TFWeb 03]

The service may choose to use the common identity (without password) as its authentication or may require the user to re-authenticate to an internal user database or the NGDS common-identity store. Data sent over the UFS interface will also be encrypted, providing assurance that it will not be compromised in transit. Using the common identity (without password) as user authentication, this methodology provides a moderate level of trust. With user re-authentication (against internal store or NGDS), this methodology provides high trust.

8. Example 5: Portal-supplied Common Identity with COTS Single Sign On (SSO) product and SSL

The portal provides common-identity information, and the UFS interface is encrypted. The service uses the common identity as its authentication, through the SSO product. Data sent over the UFS interface will also be encrypted, providing trust that it will not be compromised in transit. This methodology provides high trust.

Advantages	Disadvantages
Support for the user identity is now shifted away from the application/service developer. The application owner no longer needs to manage user passwords, but must still manage users for means of authorization.	Existing applications have to map the common identity to the existing user names. Application local user information will need to be stored in a local database.
Passwords are never sent over the UFS interface.	Requires a DoD server certificate.
Uses the portal login for application authentication. Eliminates multiple login screens.	Lower performance.

Figure 86. Advantages and Disadvantages of using a Portal-supplied common identity with COTS Single Sign On (SSO) product and SSL to secure a UFS interface. [TFWeb 03]

The service uses the common identity as its authentication, through the SSO product. Data sent over the UFS interface will also be encrypted, providing trust that it will not be compromised in transit. This methodology provides a high level of trust.

THIS PAGE INTENTIONALLY LEFT BLANK

REFERENCES

[ACM 95] "Reflections on Trusting Trust - *Ken Thompson*," September 1995. Reprinted from *Communication of the ACM*, Vol. 27, No. 8, August 1984, pp. 761-763. Copyright © 1984, Association for Computing Machinery, Inc. Available at http://www.acm.org/classics/sep95.

[ACM 03] *ACM Queue*, vol. 1, no. 8, November 2003.

[Adams 02] Adams, D.J., *Programming Jabber*, O'Reilly & Associates, Sevastopol California, 2002.

[Boyd 03] Boyd, F., "RE: XMLizing chat," e-mail, July 2003.

[Brutzman 02] Brutzman, Don, *"X3D-Edit Authoring Tool for Extensible 3D (X3D) Graphics,"* 2002. Available at http://www.web3d.org/TaskGroups/x3d/translation/X3D-EditAuthoringTool.pdf

[Brutzman 03] Brutzman, Don, "Web-Based 3D Graphics Rendering of Dynamic Deformation Structures in Large-Scale Distributed Simulations," 30 November 2003 Available at http://web.nps.navy.mil/~brutzman/DeformableSurfacesTechnicalReport2003November.pdf

[Brutzman 04] Brutzman, Don, "*Graphics Internetworking: Bottlenecks and Breakthroughs,*" January 2004. Available at http://web.nps.navy.mil/~brutzman/vrml/breakthroughs.html

[Brutzman, McGregor 03] Brutzman, Don and McGregor, Don, "XML Binary Serialization using Cross-Format Schema Protocol (XFSP) and XML Compression Considerations for Extensible 3D (X3D) Graphics," 28 September 2003. Available at http://www.w3.org/2003/08/binary-interchange-workshop/

40-BrutzmanXmlBinarySerializationUsingXfspW3cWorkshopSeptember2003.pdf

[Brutzman, Zyda 02] Brutzman, Don and Zyda, Michael, *Extensible Modeling and Simulation Framework (XMSF) Challenges for Web-Based Modeling and Simulation*, 22 October 2002. Available at http://www.movesinstitute.org/xmsf/XmsfWorkshopSymposiumReportOctober2002.pdf

[Client 03] Jabber Software Foundation, “Jabber Windows Clients.” December 2003. Available at http://www.jabber.org/user/clientlist.php

[DODD 8501] DoD Directive 8501, *Information Assurance*, 24 October 2002. Available at https://infosec.navy.mil/Documents/doc?type=dodd&tab=dod

[DODI 5200.40] DoD Instruction 5200.40, *DoD Information Technology Security Certification and Accreditation Process (DITSCAP)*, 30 December 1997. Available at https://infosec.navy.mil/pub/docs/documents/dod/dod/i520040p.pdf

[Dreamtech 02] Dream Tech Software Team, *Instant Messaging Systems*, Wiley Publishing, 2002.

[Houlihan 03] Houlihan, B., “Not Your Normal Chat Room: IWO JIMA Revolutionizes Battle Communications,” USS IWO JIMA Public Affairs Web Page, 07 April 2003. Available at http://www.iwo-jima.navy.mil

[Hunter 01] Hunter, David, et al. *Beginning XML,* 2nd Edition, Wrox Press 2001.

[Jabber 03] “Jabber Users Guide.” February 2003. Available at www.jabber.org.

[Jara 03] Jara, Timothy, and Matt Lisowski. "Don't Silence Navy Chat." *Proceedings* September 2003, pp. 52-55.

[Lee 02] Lee, S. and Smelser, T., *Jabber Programming*, M&T Books, New York New York, 2002.

[Macedonia, Brutzman 94] Macedonia, M. R. and Brutzman, D. P., “MBone Provides Audio and Video Across the Internet,” *IEEE Computer*, vol.27 no.4, April 1994, pp. 30-36. Available at http://www-itg.lbl.gov/mbone/Macedonia.html

[Macedonia 95] Macedonia, Michael R., Zyda, Michael J., Pratt, David R., Brutzman, Donald P. and Barham, Paul T., "Exploiting Reality with Multicast Groups: A Network Architecture for Large-Scale Virtual Environments," *IEEE Computer Graphics and Applications*, vol. 15 no. 5, September 1995, pp. 38-45. Available at http://movesinstitute.org/~zyda/pubs/IEEECGA95.pdf

[McCormack 03] McCormack, D., "NUWC's Chat Support Gets Thumbs Up From the Fleet," *NUWSCOPE*, laboratory newsletter, May 2003, page 3. Also provided as Appendix D.

[Monson 01] Monson-Haefel, R. and Chappell, D., *Java Message Service*, O'Reilly & Associates, Sebastopol California, 2001.

[MacVittie 03] MacVittie, Lori, "Manage VPN Services The Survivor's Guise 2004: Business Applications" Network Computing, 16 December 2003. Available at http://www.nwc.com/showitem.jhtml?articleID=17000088&pgno=5

[Neushul 03] Neushul, James D., "Interoperability, Data Control and Battlespace Visualization Using XML, XSLT AND X3D," Masters Thesis, Naval Postgraduate School, Monterey California, September 2003. Available at

http://www.movesinstitute.org/Theses/Neushulthesis.pdf

[Oikarinen 00] Oikarinen, Jarkko, "IRC History" May 2000. Available at http://www.irc.org/history_docs/jarkko.html.

[Perkins 03] Perkins, Phillip, "Send XML without XMLHTTP," Originally published in the Builder.com Web Development Zone e-newsletter, 1 April 2003. Available at http://builder.com.com/5100-6371-5034602.html#Listing%20A

[Poul 02] Poul, Gerhard, "Instant Messaging for E-Business," 1 May 2002. Available at www-106.ibm.com/developerworks/wireless/library/wi-jabber.

[Policies 98] Department of Defense, *Web Site Administration Policies & Procedures,* Section V, 25 November 1998, Available at http://www.defenselink.mil/webmasters

[Pullen 99] Pullen, Mark, "Reliable Multicast network Transport for Distributed Virtual Simulation, "*Proceedings of the 1999 IEEE Distributed Interactive Simulation and Real-Time Systems Workshop*", Available at http://netlab.gmu.edu/pubs/pullen-disrt99.pdf

[Ritter 03] Ritter, Paul, “Instant Messaging Gains Ground with Businesses and Consumers,” September 9, 2003. Available at http://techupdate.zdnet.com/techupdate/stories/main/0,14179,2914639,00.html

[Rose 03] Rose, Kevin, “Dark Tip: AIM Sniff,” July 21, 2003. Available at http://www.techtv.com/screensavers/answerstips/story/0,24330,3482002,00.html

[Schacher 03] Schacher, G., Pilnick, S., Irvine, N., and Gallop, S. “Seventh Fleet Field Training Exercise Fleet Battle Experiment Kilo Fires Initiatives Final Report,” Technical Paper, Naval Postgraduate School, Monterey California, August 2003, page 116.

[Shigeoka 02] Shigeoka, I., *Instant Messaging in Java*. Manning Publishing Co., Greenwich Connecticut, 2002.

[Studio] Jabber Studio resource website, 2003. Available at http://www.jabberstudio.org

[Support 02] Jabber Support, “General Jabber FAQs” Jabber Inc. 2002. Available at http://support.jabber.com/faqs/jabberfaq/general.html

[Sullivan 03] Sullivan, Paul, RADM USN, “Sub Director Foresees ‘Revolutionary’ Power of SSGNs,” *Sea Power*, July 2003. Available at http://www.navyleague.org/sea_power/jul_03_21.php

[Sutton 03] Sutton, David, (Peregrine) “GroupChat problem – Error 501,” bulletin board post, August 2003. Available at http://jabber.open.ac.uk/BB-forum/viewforum.php?f=5

[TFWeb 03] Task Force Web, “Navy Enterprise Application Development Guide,” Version 1.12, 10 February 2003.

[Thomas 03] Thomas, Jay. "Don't Silence Navy Chat." *Proceedings* December 2003, pp. 18-22.

[Trinque 03] Trinque, Derek. "Don't Silence Navy Chat." *Proceedings* November 2003, pp. 18-22.

[Virtual 03] VirtualTEAMWORKS.com, Resource Website, 2003. Available at: http://www.virtualteamworks.com/03.htm

[Webopedia] Webopedia, Resource Website, 2003. Available at http://www.webopedia.com/TERM/c/chat.html

[Wheeler 03] Wheeler, David, "Why Open Source Software / Free Software (OSS/FS)? Look at the Numbers!," Resource Website, September 2003. Available at: http://www.dwheeler.com/oss_fs_why.html

THIS PAGE INTENTIONALLY LEFT BLANK

INITIAL DISTRIBUTION LIST

CNO Strategic Studies Group (SSG)

Defense Technical Information Center
Ft. Belvoir, Virginia

Dean of Research, NPS

DON CIO XML Working Group

Dudley Knox Library
Naval Postgraduate School
Monterey, California

MOVES Institute Advisory Board

NFOBAN mailing list

Savage announcements email list:
savage-announce@MovesInstitute.org
Subscription information available at http://www.MovesInstitute.org/mailman/listinfo

US Naval Institute

Web3D Consortium members

XMSF Partners announcements email list:
xmsf-announce@MovesInstitute.org
Subscription information available at http://www.MovesInstitute.org/mailman/listinfo

LCDR Edwin Armistead USN, OPNAV 09WN3A

David Bellino, NUWC

RADM David Bill USN Ret.

Dan Boger, NPS

Alex Bordetsky, NPS

Anthony Cerri, USJFCOM

Erik Chaum, NUWC

Frank Cooper, Concurrent Technologies Corporation (CTC)

Mark Crawford, DON CIO

John Davis, UMASS Dartmouth

Bruce Delagi, Sun

Peter Denning, NPS

RADM Tom Fellin USN Ret.

Shelley Gallup, NPS

CAPT Gunderson USN
Commander, FNMOC

Bob Hartell, USA CIV MP REP
Combined Arms Command
Ft Leavenworth, Maneuver Control System (MCS)

CDR Sue Higgins USN Ret., NPS

Scott Hoffman, J6I SADI P/M

CAPT Wayne Hughes USN Ret., NPS

Michael Jacobs, DON CIO

CDR Timothy Jara USN

CAPT Jeff Kline USN, NPS

Dick Lee, OSD C4I ACTD

LCDR Matt Lisowski USN

Francisco Loaiza, IDA

Bowen Loftin, Old Dominion University (ODU) VMASC

CAPT Arnie Lotring USN
Commander, Submarine Learning Center

VADM Richard W. Mayo USN Ret.

Donald McCormack, NUWC

Jim McGuiness, Sun

LT Jim Mcreynolds USN, OPNAV N70

CAPT John Mickey USN, NUWC

Katherine Morris, SAIC

Dr. Richard Nadolink, NUWC

Greg Papadopoulos, Sun

Mark Philips, Old Dominion University (ODU) VMASC

LCDR Phil Pournelle USN, OPNAV N81

Mark Pullen, George Mason University (GMU)

Monica Shepard, Task Force Web (TFW)

Gene Simaitis, IDA

RADM Andrew M. Singer USN
Commander, NETWARCOM

K.C. Stangl, NAVAIR

CDR Scott Starsman USN, OPNAV 09WN5

RADM Paul F. Sullivan USN
Commander, COMSUBPAC

Dr. Ahn Ta, MITRE

CAPT Jay Thomas USN Reserves

Adreas Tolk, Old Dominion University (ODU) VMASC

LCDR Derek Trinque USN

William J. (Joe) Welch
Space Systems Academic Group (SSAG)

Ken Williams, JFCOM Dept J86

CDR Chris Vance USN, NPS

Mr. David M. Wennergren, DON CIO

Mike Zyda, NPS