ROOS: A Structured Email Server
by
Igor L. Belakovskiy
S.B., Massachusetts Institute of Technology (2001)
Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of Master of Engineering in Electrical Engineering and Computer Science at the MASSACHUSETTS INSTITUTE OF TECHNOLOGY
May 23, 2002
© Igor L. Belakovskiy, MMII. All rights reserved.
The author hereby grants to MIT permission to reproduce and distribute publicly paper and electronic copies of this thesis document in whole or in part.

Author .........................
Department of Electrical Engineering and Computer Science
May 22, 2002

Certified by ....................
Tomas Lozano-Perez
Cecil H. Green Professor of Computer Science and Engineering
Thesis Supervisor

Certified by ....................
Scott Boag
Senior Software Engineer, Advanced Technologies Group, International Business Machines Corporation
Thesis Supervisor

Accepted by ............
Arthur C. Smith
Chairman, Department Committee on Graduate Students
ROOS: A Structured Email Server

by

Igor L. Belakovskiy

Submitted to the Department of Electrical Engineering and Computer Science on May 23, 2002, in partial fulfillment of the requirements for the degree of Master of Engineering in Electrical Engineering and Computer Science

Abstract

In this thesis, I discuss the design and implementation of an XML Schema based electronic mail discussion system, known as the Rules of Order Server (ROOS). This project is aimed at improving the quality of interactions in electronic mail discussions, involving a large number of participants and contributions. The goal was to add structure to discussions in order to achieve better discussion flow, as well as more as a more focused discussion environment. The project was implemented in Java and utilizes XML, XML Schema and XSL standards issued by the World Wide Web Consortium.

Thesis Supervisor: Tomas Lozano-Perez
Title: Cecil H. Green Professor of Computer Science and Engineering

Thesis Supervisor: Scott Boag
Title: Senior Software Engineer, Advanced Technologies Group, International Business Machines Corporation
Acknowledgments

First, I would like to thank Scott Boag for being a great mentor and introducing me to cutting-edge technologies and new ways of using them. Without his vision, this project would have never existed.

I would also like to thank Emily Farmer for being a great manager and helping me throughout both of my co-op assignments with her group. I would like to thank Shane Curcuru, Paul Dick, Don Leslie, Joe Kesselman, Myriam Midy, Dave Marston and Dave Bertoni for providing feedback about ROOS, as well as their outstanding technical advice. Bob Fiorenza and Steve Rohall deserve my gratitude for first bringing me to IBM, and later to the Advanced Technologies Group.

I would like to thank my thesis advisor, Professor Tomas Lozano-Perez, whose insights and expertise proved invaluable during the development as well as the writing process. Professor Albert R. Meyer has been my academic advisor for almost 4 years, and his knowledge and advice have been an integral part of my MIT experience. I would also like to thank Professor Steven B. Leeb for graciously allowing me to be a teaching assistant for his Microcomputer Project Lab, so that I could support myself while I was writing this document.

I would like to thank my family, for without their continuing support, none of this would be possible. My father Leonid, mother Zoya and brother Aleksander helped me to developed both as a student and as a human being. They have my eternal gratitude.

Finally, I would like to thank all of my friends for providing me with data, proof reading help and general support during this process. In particular, I'd like to thank my girlfriend, Julie Zeskind, for always being being there for me when I needed help. I would also like to thank my friend Eugene Weinstein for proof reading this document.
Contents

1 Introduction 13
   1.1 Problems ..................................................... 15
      1.1.1 Volume ................................................. 16
      1.1.2 Cohesiveness ........................................... 16
      1.1.3 Archiving and Retrieval .............................. 17
   1.2 ROOS ............................................................. 18
   1.3 Limitations .................................................. 19
   1.4 Outline ....................................................... 20

2 Background 21
   2.1 Related Work in the Field ................................. 22
      2.1.1 Babble .................................................. 22
      2.1.2 Bulletin Boards ....................................... 23
      2.1.3 Reinventing Email .................................... 24
      2.1.4 Lotus Workflow ....................................... 24
      2.1.5 Other Systems ......................................... 25

3 How ROOS Works 29
   3.1 ROOS Users .................................................. 29
   3.2 Discussion Initiation ...................................... 29
   3.3 Discussion Participation ................................. 31
      3.3.1 Special Messages ..................................... 32
      3.3.2 Discussion Navigation ............................... 32
List of Figures

1-1 Daily Mail Traffic On Xalan-Dev. .......................... 14
1-2 Active Threads Per Day on Xalan-Dev. .................... 15
2-1 Thread Visualization of the Reinventing Email Client. Reprinted with permission from [19] .......................... 25
2-2 The Search Interface of the Reinventing Email Client. Reprinted with permission from [19] .......................... 27
3-1 The ROOS Login Screen ........................................ 30
3-2 Email Message:Navigation Detail. Underlined items indicate HTTP and mailto links. .......................... 33
3-3 WWW Message Detail. ........................................ 34
3-4 A ROOS Invitation Message ................................. 36
3-5 Email Message:Transition Table ........................... 37
3-6 Email Message:Invitees ....................................... 38
3-7 The ROOS Discussion Initiation page. ....................... 39
3-8 A WWW View of a ROOS Discussion. ....................... 40
3-9 Viewing a Specific ROOS Message. ......................... 41
4-1 Discussion Component Definition ............................ 44
4-2 componentType Extension Examples ........................ 45
4-3 Grammar Equivalent of Schema Definitions. .............. 46
4-4 Legal Instances of Figure 4-2 Type Definitions........... 46
4-5 Minimal Discussion Document ............................... 48
4-6 Typical Entry In A Legal Actions File

4-7 Typical Entry In The Tag Descriptions File

4-8 The Configuration Module

4-9 The Discussion Module

4-10 The Presentation Module
## List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>B.1</td>
<td>Xalan Dev Traffic Analysis</td>
<td>79</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

Ever since the first electronic mail application was created in 1971[2], e-mail has been the driving force behind many technological and social changes. While the creator himself may not have realized what he had created, others soon began to recognize “that the techniques of network mail are going to sweep the country and drastically change the techniques used for intercommunication in the public and private sectors” [2]. E-mail has become a communication tool whose importance rivals that of the telephone, and whose use spreads further and further every day.

As the use of e-mail becomes increasingly pervasive in society, the complexity of tasks performed by email increases as well. Whereas ten years ago, e-mail was just that—an electronic version of regular letters, today, it is used to make decisions, in lieu of formal meetings, design products, and many other forms of collaboration previously unthought of. Many open source and standards groups, such as the Apache Software Foundation (Apache) and the World Wide Web Consortium (W3C) use e-mail as their primary communication medium. Apache uses electronic communication for tasks ranging from problem reports, to announcements of new features, and to the actual design of new features. Frequently, radical architecture changes, and new product development is discussed over email as well. The diversity of tasks, however, results in a veritable e-mail flood. Apache attempts to deal with the problem by creating project specific mailing lists, and creating separate developer and user lists within each project. For example, the Xalan[8] project has at least four mailing lists—two
user lists, and two developer lists. The two user lists, xalan-j-users and xalan-c-users, are for general usage questions for the Java and C versions of Xalan, respectively. Furthermore, there are two developer lists, one on which advanced usage, and all development questions are discussed (xalan-dev), and one to notify developers about modifications to the Xalan CVS repository (xalan-cvs). This reduces the amount of traffic per list; yet, on average, members of the developers discussion list still receive over 20 messages per day, with peak days topping 40 messages (see figure 1-1). These messages range from complex software flaws and design discussions to full-fledged policy discussions about the direction of the Xalan project, or a particular sub-project within it. On average, there are almost 10 new threads of conversation per day, with length varying from 1 to 20 messages (see figure 1-2, and Appendix B). With such a diversity of traffic, attempts to lead a focused discussion often fail, increasing the frustration level of most members of the Xalan community. This failure is certainly
not limited to Xalan, and indeed, issues brought up by Xalan users are echoed by most members of on-going collaborative efforts over e-mail.

![Active Threads Per Day](image)

Figure 1-2: Active Threads Per Day on Xalan-Dev.

1.1 Problems

Trying to start a prolonged and focused discussion over a high traffic electronic mailing list is very similar to trying to start a organized forum about world peace at a cocktail party. You have to hope that your attempts at starting the discussion are heard over all of the noise, you wish that that annoying man would stop reminiscing about his days in the Peace Corps, and you hope that you remember anything that is said the next day. In other words, your attempts at starting a discussion may not be taken up by others, who are lost in all of the traffic on list; users that do get involved in the discussion tend to pursue tangents not at all relevant or interesting to you;
and finally, even if a high-volume discussion is conducted, it's usually unusable for reference purposes.

1.1.1 Volume

The difficulty in starting a long term discussion within a high traffic mailing lists mainly lies in the fact that most users have great difficulty just reading their e-mail. In fact, some users report spending as many as ten hours at a time on reading through their e-mail[23]. In most cases, the task of reading incoming e-mail probably takes closer to one to two hours daily. Getting involved in a high intensity discussion over email further increases the amount of time devoted to reading. In the case of the Apache community, the situation is exacerbated by the fact that many Apache contributors happen to also have day jobs that may not involve their Apache project, resulting in most people being very reticent to involve themselves in a large scale electronic discussion. In addition to their regular parsing of incoming messages, the users involved in the discussion have to pay special attention to messages which pertain to the discussion. Those that do not, and attempt to participate are quickly discounted as uninformed, and are often “flamed”\(^1\). Perhaps the best way to deal with this problem is to separate long discussion threads from the every day short threads that dominate most mailing lists.(see table B in Appendix B). However, even if that is done, other problems will arise.

1.1.2 Cohesiveness

The most common complaint heard from participants in electronic collaboration is the difficulty that lies in enforcing a coherent direction on an e-mail discussion. Indeed, most discussions inevitably end up pursuing tangential interests of various members, instead of following the path intended by the originator of the discussion. Even if a satisfactory conclusion is reached eventually, it is reached by pursuing many irrelevant issues, and expending the energy of the participants on matters that could have well

\(^1\)Verbally attacked over e-mail
been ignored, without any detriment to the discussion.

Imagine the United States House of Representatives discussing a resolution against Iraq, then following a chance remark to discuss the Israeli-Palestinian conflict, and then trying to decide on a resolution against Iraq again. Such a session would probably last about a week, and probably result in a dead end. Clearly, this is unacceptable in the legislative process. Just as clearly, this should not be acceptable in an electronic discussion, whether over e-mail or in real time.

It must be recognized, however, that simply laying down guidelines for discussions is not enough. Users repeatedly violate guidelines that are not strictly enforced—replying to messages marked “DO NOT REPLY DIRECTLY”, inserting their comments when a vote is called for, etc.. Clearly, a moderator is required. Yet, moderating a prolonged discussion can be very time consuming, and most users would rather avoid discussions altogether than have to moderate one. Aside from hiring someone as a full time moderator, what could be done? So far, very little has been done, resulting in the increasing level of frustration of the participants in online collaboration.

1.1.3 Archiving and Retrieval

Another source of irritation is the difficulty most people have in using archived discussions, whether in their own mailboxes, or, as is the case with most large lists, through the World Wide Web archives of correspondence. In the case of the mailbox archive, important messages frequently end up getting erased or misplaced. Short of not deleting any messages, there really is no cure for the former problem, except for the afore-mentioned web archives. In addition, most users find it inherently difficult to file their messages. As a result, some messages may end up in a folder very different from the one the user expects to see them in. This in turn increases the time it takes to find a relevant message, causing further aggravation.

It would appear that the answer to mailbox archive flaws lies on the web. Web archives store all the messages from a list without deleting them and the messages are always kept in the same format. However, even navigating to the correct discussion thread is a fairly cumbersome process. Once the thread is located, the problems with
a web archive become even more apparent.

It is not enough to merely find a discussion thread. In order to make use of it, one must also be able to easily find the important messages from any particular discussion. Six months after a major design has been decided upon, no one will be concerned about a set of features that has been long discontinued, yet the discussion about those features is hard to filter from the rest of the thread. Simply put, over one hundred messages that have the same subject line, are extremely tedious to search through. A textual search is only useful if one remembers the phrases used in a particular message. Without that knowledge, the task of finding the right message remains tedious. For a new participant in the collaborative effort, the task is nearly impossible without assistance by more experienced users. For both old and new members alike, there is a significant time drain associated with knowledge acquisition from mailing list archives.

Finally, web based archives offer no insight into the work flow of the discussion beyond the thread structure of the discussion. There are no indicators to signify that a message was intended as question, a comment, or a resolution.

1.2 ROOS

The goal of this project is to solve at least some of the problems mentioned in the previous section. The Rules of Order Server (ROOS) will allow users to separate large discussion threads from everyday mailing list traffic, to define the rules for their discussions and to be able to quickly find messages in an archived thread.

ROOS provides a quick and easy interface for users to initiate their own discussion. By initiating a discussion, the user essentially creates a special purpose mailing list, which is governed by a set of rules selected or created by the user, and whose objectives are stated in the discussion proposal, written by the originator or “author” of the discussion. These rules define the structure of the discussion, such as how different types of messages interact with each other, and the appearance of a completed discussion, as well as the length of the discussion, and individual sections
within it. Furthermore, users invited to participate in the discussion may choose not to do so, by unsubscribing from the particular discussion, without losing contact with the mailing list that the discussion originated from.

The rule set defined by the author acts as a virtual discussion moderator, permitting certain messages only at times when they are appropriate. For example, if a user is responding to a call to vote, he must state his position on the subject of the vote, and will not be permitted to simply make a comment as to how much he or she enjoyed the current issue of WIRED\(^2\). The rule set also permits the author to define deadlines for various parts of the discussion, resulting not only in a focused, but also a timely discourse.

Finally, since messages are differentiated by type in the rules of discussion order, they can also be tagged by their various types after they have been accepted by the virtual moderator component of ROOS. This tagging allows the users to not only differentiate message types in real time, but even more importantly, to be able to easily locate sections of the discussion months after the discussion has been completed. Each discussion is preserved, along with its rules on the server, and can be recalled at any time. The relevance of this can be seen in many open source projects, where new members join the effort frequently, and are frequently refereed to list archives for major design discussions. Tagging would not only make it easier for a new user to follow a design discussion, it would also reduce the drain on the resources of those already familiar with the discussion, allowing them to work on the project, rather than familiarizing new users with past decisions, designs and policies.

### 1.3 Limitations

It is important to note that ROOS is not intended to solve all problems associated with high volume mailing lists. Certainly, ROOS would reduce the traffic volume on large mailing lists by taking discussions off the main mailing list. However, allowing long discussions to be conducted in a focused environment may actually increase the

---

\(^2\)This type of aside occurs with astonishing frequency on mailing lists
total amount of messages for users involved in the discussion\textsuperscript{3}. The slight increase in message volume could actually be beneficial to the discussion, as it keeps the users more aware of the discussion flow.

ROOS is an enhancement on the mail server side of the e-mail process. It will not improve every aspect of the email experience for the users, just the part that deals with large discussions. Filing messages generated by ROOS in a reasonably manner is still the responsibility of the individual users. It must be noted, however, that work is being done at IBM Research that aims at improving the e-mail client, in particular from the conversation thread perspective. Since all ROOS discussions are easily identifiable threads, any thread-aware program could potentially be able to take advantage of this. Before this could be done, however, standard e-mail headers may need to be expanded to enable thread recognition. It is my hope that IBM Research will be the driving force behind this expansion in the coming years.

\subsection*{1.4 Outline}

Chapter Two discusses the origins of the project and relevant research in the area of electronic collaboration. Chapter Three covers the users' interactions with ROOS. Chapter Four discusses the design and architecture of ROOS. Chapter Five discusses issues that arose during the implementation phase. Finally Chapter Six summarizes the project, with results, conclusions and future improvements that could be made to ROOS.

\textsuperscript{3}This was shown to be the case in testing. It is difficult to say whether or not the increase in traffic can be attributed solely to efficiency. Some was most certainly due to the novelty value of the system. Over time, traffic settled down, remaining at a slightly higher level than before the introduction of the system.
Chapter 2

Background

The concept of ROOS is the brainchild of Scott Boag, Chief Architect of the Xalan project in the Advanced Technologies Group at IBM Research. Over the past few years, he has been heavily involved in many electronic collaborative efforts. He has been an active member of the Extensible Stylesheet Language (XSL) work group at the World Wide Web Consortium. He has also been heavily involved with various Apache Software Foundation (ASF) efforts, ever since the Xalan code base was donated by IBM to the ASF as an open source project.

Scott’s increasing frustration with the lack of focus of the e-mail components of these efforts led him to develop the concept behind ROOS. He believed that a set of structured rules might be applied to electronic communication. Robert’s Rules of Order[18] would be an extreme example of such a rule set. This added structure could then lead to consensus, or at least acceptable resolutions on most issues. By adding deadlines, and allowing the user to customize the discussion rules, it would be possible to decrease the amount of time spent on participating in the discussion, and at the same time, improve the quality of the discourse taking place. ROOS is only one of the many attempts at improving the quality of on line collaboration, and the remainder of this chapter is devoted to some of those efforts.
2.1 Related Work in the Field.

Various efforts have been made to supplement or even supplant traditional electronic mailing lists and/or change the user’s email experience. This section will discuss some of these.

2.1.1 Babble

Perhaps the closest to ROOS in its goals is the Babble project at the Social Computing Group at IBM’s Thomas J. Watson Research Center[1]. Babble aims at improving collaboration by creating a richer environment for on-line interaction. For instance, Babble uses a circle to represent a “conversation”, and dots to represent users. The proximity of the dots to the center of the circle indicates the level of a user’s activity in a conversation. Babble attempts to do away with the constraints of email, by presenting all contributions to a particular “conversation” in one view, thus avoiding appending email histories to each new contribution, in order to keep the contribution in context. This particular aspect of Babble is very similar to ROOS, which only presents a single message to the user, and provides an easy interface for navigating to other parts of the discussion (see section 3.3.2).

Unlike ROOS, Babble does not currently offer a way to create a discussion flow within a Babble conversation. Instead, Babble relies on the informality of the overall environment to allow users to interact with each other quickly. Users quickly gain access to information by skipping most of the formal e-mail etiquette, and simply asking a question—e.g.—“Hey John, do you know what version Xerces are we using?”.

It would seem that such system would be better suited for a group of people the members of which are fairly familiar with each other, such as a product group within a large corporation; or a fairly informal gathering of friends that are looking, to do just what the system name suggests—babble. In a larger, more diverse community, where at least some of the users don’t interact on a daily basis, in order to achieve a more focused discussion, some structure should be introduced. This does not mean that Babble has no place in a large collaborative effort. In fact, for simple, day to
day questions and answers, Babble seems to be a better way to communicate than a typical mailing list. Perhaps, Babble and ROOS can change the world of online collaboration together.

2.1.2 Bulletin Boards

Electronic bulletin boards provide another way to conduct lengthy discussions. However, these usually become very close to unusable without active moderation. While the content is presented in one place, thus avoiding the need for history, most bulletin boards without moderators sooner or later are buried in a flood of personal attacks, off-topic remarks, etc. The complete lack of structure, and moderation produces results that are even more unusable than a typical mailing list. Of course, moderated bulletin boards are far more polite – moderators usually remove most personal attacks, and attempt to force users to stay on topic. However, in a project where most of the users have full time jobs not necessarily devoted to the project itself, very few people would volunteer to moderate a discussion. Impartial moderation is very difficult to find, and members that do end up moderating discussions may find that their editorial skill (or lack thereof) alienates many participants. Certainly, starting a focused discussion on a bulletin board requires a great deal of effort.

Content labeling on a bulletin board is usually not very advanced. Most boards simply have a list of ongoing discussions, and discussion contributions are simply displayed in the order in which they were sent. This offers no real advantage over a standard mail client, and makes it very difficult to understand hows posts are related to each other, unless the writer explicitly states which post he is responding to. Essentially, bulletin boards were designed for very unstructured conversation, and while they remain an effective medium for that task, they do not appear to be suited to focused and easily understandable discussions.
2.1.3 Reinventing Email

The Reinventing Email project[15, 19] is aimed at improving the user email experience through the use of a more advanced email client. The new email client allows users to visualize conventional email threads in their mailbox, as well as modifying them by adding new elements as they deem fit. This allows users to categorize messages according to their perceived relations to each other, and not merely their general topics, in the manner of the traditional folder filing systems. The system also allows the user to visualize the messages in a thread through a time-line interface (see figure 2-1). A search feature with a calendar style layout allows the user to get a day by day summary of all messages matching the search criteria (see figure 2-2). Overall, the client greatly enhances the user experience with email.

The project, while certainly offering at least a partial solution to the archiving problem, does not really solve the lack of focus problem. The client does a great deal to reduce the chaos in the user’s mail box after messages arrive there, however, operating on the client side prevents the system from actively attempting to reduce or streamline the traffic into a mailbox. Thus, a system like ROOS would still be of great benefit to the client’s user, in fact, due to the built-in thread information in a ROOS message, the two systems would complement each other very well.

2.1.4 Lotus Workflow

Lotus Workflow is an application aimed “at streamlining time-consuming, people-based activities, and tracking their progress”[4]. The Workflow application allows developers and business managers to assign, schedule, and track each step of a business process. Workflow provides the structure needed to run any business process, no matter how complicated. It is a very powerful application, that can add structure to most business processes. At any enterprise using Notes, deploying Workflow is fairly quick, and can be easily integrated into the every day lives of the employees.

However, the dependence of Workflow of Notes makes it somewhat less practical in a collaborative effort that might be based in different parts of the world, at different
companies, and very frequently on diverse platforms, as most open source efforts are. Most participants in the efforts have grown accustomed to their preferred email interface, and do not desire to use a separate new interface to participate in, what for most of them is essentially a secondary job, or even a hobby. In the case of most open source efforts, budgets are limited, and deploying a whole new email and productivity system is simply not an option. Workflow is a great solution for a conventional company or corporation, however, it simply would not work for such a diverse group of people as the open source community.

2.1.5 Other Systems

There are many more systems aimed at improving Internet collaboration. The Sun Microsystems SunForum Application[14] is a complete collaboration solution, including white-boarding, file sharing, and video conferencing. Red Hat Incorporated provides ECM (Enterprise Collaboration Management)[17], which focuses on knowledge sharing. All of these systems are extremely powerful, and can be very effective when used and supported properly. Unfortunately, the open source community, it simply does not have the resources required to deploy and support these systems.
Furthermore, many functions of these systems, such as video conferencing are not very integral to day-to-day open source collaboration, and others, such as file sharing are already provided by such systems as the Concurrent Version System (CVS), the dominant open-source version control system. Full fledged real time collaboration is often unnecessary, or even impossible, as contributors operate in different time zones and on very different schedules. Most contributors are satisfied with their current methods of communication, and some may want to improve efficiency. However, just about all are unwilling to stop using email as their primary means of communication.
Chapter 3

How ROOS Works

ROOS allows the user to create his own discussion, moderates the discussion, and creates persistent storage for the discussion even after it has been concluded. This chapter will discuss the ROOS user interface, and users’ interactions with it.\footnote{The details of how and why certain features were implemented will be not be covered in this chapter. An in-depth look at the technical details of the UI can be found in chapter 5.}

3.1 ROOS Users

To participate, or even create a new ROOS discussion, a user does not have to be registered with ROOS. However, for the users’ convenience, an account system was created. This allows users to take advantage of features such as a personal address book, thus making the discussion initialization process faster for frequent users. In the future, this feature will also allow users to keep track of the discussions they are involved in, and may even evolve into the primary interface to ROOS.

3.2 Discussion Initiation

To initiate a discussion, the user must either login to his account, in order to use his personal address book, or login as a guest user and use a default address book (see figure 3-1). In either case, the address book merely serves to accelerate the process
of selecting participants for a new discussion, and other participants can be added as well. After the user logs in, and chooses to start a new discussion, he is taken to the discussion initiation page (Figure 3-7.) The discussion initialization page allows the user to create a working title for the discussion, write a discussion proposal, decide whether or not the invitation list is restricted, select an existing discussion template\(^2\) or provide the HTTP address of their own template, and input an invitation list for the discussion\(^3\). Users not in the address book can be added to a particular discussion, allowing users to keep a relatively small personal address book.

ROOS provides several default discussion templates, ranging from technical discussion to town hall meeting. These templates are fairly generic, and contain a large number of optional elements, allowing greater flexibility to users, while still retaining a notion of structure. New templates can be easily created as well, using any XML\(^{[11]}\) or XML Schema\(^{[10]}\) editor, or even a text editor\(^4\), though it is hoped that only advanced discussion users will attempt to do so. This is mainly due to the fact that a poorly designed discussion will not succeed regardless of the technology behind it. A user creating a new discussion type should spend a fair amount of time on dis-

\(^2\) or discussion rule sets, as discussion in section 1.2

\(^3\) If a user is not using a personal account, he will also have to identify himself; otherwise, the initiator information is filled in automatically

\(^4\) see section 6.3.2 for future improvements in this area
discussion design before implementing a template for it. The user can select a template by clicking one of the radio buttons, or typing in a URL to his own template.

Once the discussion template is chosen, the invitation list is created and other relevant information has been entered, the user is asked to enter the duration of the discussion. Once the duration is set, the discussion initiation process is completed, and invitations are sent out to all participants.

3.3 Discussion Participation

An invitation to participate in a discussion contains the discussion proposal, instructions on how to participate in the discussion (Figure 3-4), a legal responses table (Figure 3-5), and the list of invited users(Figure 3-6). Once an e-mail invitation is received by a user, he can begin participating in the discussion. Each discussion message contains a table of legal responses, along with a brief description of each response. This table is aimed at ensuring that each user understands what a certain response means, instead of merely clicking on the one that sounds appropriate. Once the user decides how to follow up on a particular message, he simply clicks on the appropriate response link, and a new message window is brought up in his mail client. The subject line of this new e-mail contains the type of follow up message that the user chose. The participant then composes his response, adds any attachments, if desired, and sends the message to ROOS.

If the transition chosen by the user is appropriate, and the deadline for contributions has not passed, ROOS accepts the message. Upon acceptance, the contribution is added to the discussion archive, and if the contribution type allows follow up contributions, a transition table is appended to the text of the message, and the new message is sent out to all participants in the discussion. Should a contribution violate discussion rules, or arrive after a deadline is passed, it is simply returned to the user, notifying him of the type of violation he has committed, and suggesting alternate

---

5 Besides the "global" discussion deadline, which is set when the discussion initiation begins, certain sections of the discussion may have their own deadlines, as well. For example, a vote will always have a deadline.
3.3.1 Special Messages

Certain contribution types are handled differently from the general pattern described above. For example, when a vote is cast, the message received by participants will state the position of the voter, regardless of whether or not the position is stated in the body of the message. When a motion is made by a participant, another one must second it. After the motion has been seconded, the users receive the original motion again, with an indication that it has been seconded.

Another example of a variation on the standard message scheme are messages that have a built-in deadline attribute. When using this type of message to respond, the user is given a default duration, but has the option of changing it in the subject line of his response. It is important to note that while not all discussion contribution types have an explicit deadline, they have an implicit one, derived from previous messages in the discussion thread. In other words, all contributions that follow up a message with an explicit deadline, expire when that message expires. Since the discussion itself has a deadline, ultimately, a deadline for all contributions does exist.

Besides user-generated messages, ROOS also sends out automatically generated reminders and deadline notifications. For any discussion segment that lasts longer than twelve hours, a deadline reminder is sent out twelve hours prior to the deadline. Once a deadline has passed, a deadline message is sent out. In the case of a vote, this message also contains the tally of the votes of participating members.

3.3.2 Discussion Navigation

In addition to the table of legal additions described above, each message that follows the original discussion proposal will also contain a “snapshot” of the state of the entire discussion thread (see figure 3-2). The snapshot is a text representation of the discussion tree, containing click-able HTTP links to all active sections of the discussion, as well as mailto links to potential new contributions to that particular
segment of the discussion. The HTTP links allow the users to view other messages in
the discussion thread through a web browser. In a thread-aware e-mail client, such
as the one being developed at IBM Research [15], another window would be created
within the mail client, to view the relevant message. At this time, however, no such
capability exists within regular e-mail clients, and the browser provides an adequate
substitute.

To unsubscribe from this discussion, click here

Below are all sections that are currently active:

- proposal
- issue
- action
- item
- callToVote
- data
- data
- data
- data
- discourse
- position
- For
- or
- Against
- clarifyingQuestion
- strawVote
- comment
- comment
- reply
- reply
- reply
- reply

Figure 3-2: Email Message:Navigation Detail. Underlined items indicate HTTP and
mailto links.

3.4 WWW Capabilities

When the user views the discussion through a browser, he will see the interface
pictured in Figure 3-8. The interface consists of three panels—a search panel at the
top, an index panel on the left, and a content panel on the right of the screen. The
user may navigate to any message in the discussion by clicking on the appropriate
message number in the index panel. If the discussion participant followed a link from
an email client to view a particular message, the content panel will have that message
at the top of the visible part of the discussion (see Figure 3-9\textsuperscript{6}). Otherwise, the user will see the view in Figure 3-8, which centers around general discussion information, such as participants, and the discussion proposal.

Each contribution has a different indentation level, to reflect its depth in the discussion tree. This allows the user to easily distinguish between messages belonging to the same sub-thread, and messages on the same contribution level. This allows the user to detect whether or not a particular contribution results in an extensive discourse. For example, the contribution 12, issue contribution 13, discourse and contribution 18, comment, in figure 3-9, are all part of the same sub-thread, the attachment test case referred to earlier in this section. The indentation allows to user to view a small number messages in a sub-thread at a time, and to determine the termination point of any sub-thread by simply noticing when the indentation level stops increasing.

### 3.4.1 Contribution Features

**proposal (1): Igor L Belakovsky**

This discussion is set up to demonstrate the many wonderful features of ROOS

**Actions you may take:**

- **issue**
  A problem or obstacle that might hamper the discussion
- **actionItem**
  A task that needs to be performed by an individual in order to proceed
- **callToVote**
  A call to vote on an issue, proposal or discussion item. Must include a "second" before the vote proceeds. Only one such call is allowed, per subsection of a discussion.
- **data**
  A link to any supporting data for the proposal or counter proposal that includes this contribution.

*This section and its children will expire at 10:54:00 on 12-20-2001*

Figure 3-3: WWW Message Detail.

Each contribution displayed through the web has the same basic features as a contribution viewed through the mail client. The web view has a contribution tag and number, as well as the author's name and a table of legal actions, just like the e-mail

\textsuperscript{6}In this particular example, I wanted to see if my file attachment system worked properly. I followed a link in my mail client to see if the attached file was available on the web.
version (see Figure 3-3). The bottom of the message also contains its expiration date, in bold if the message is still active, and in a regular font if the message has already expired. In the latter case, the text of the contribution is rendered in light gray, to give the user a visual cue that the part of the discussion is no longer active. This is an obvious enhancement of the e-mail client view, since under normal circumstances, all messages are current when they are received by the mail client. A generic mail client also has no way of knowing the message is expired or not, as there is no standard way to communicate that information. In the WWW view, however, the presentation server knows exactly when messages expire, and can enhance the user experience accordingly.

A further enhancement offered by the web view is the color categorization of any contribution type that requires a yes or no answer, for example a vote, or a position. Depending on whether or not one supports a particular proposal, his response would be rendered as red or green. This allows to users to focus on those opposed or those supporting a message, whether to try and see if someone had a valid objection in a past design discussion, or to attempt to build a coalition in an electronic town hall meeting. This feature is also available through the mail client, though as merely an addition to the subject line, and not as a color coded view. Those users who prefer visual cues would probably prefer the WWW presentation of the discussion over the e-mail client version.

---

7 Assuming the user checks his email reasonably frequently, and that deadlines are of reasonable length–i.e. longer than the lag time of the mail server.
8 If you don’t cast a vote, it is assumed that you abstained.
Figure 3-4: A ROOS Invitation Message

Proposal:
This is a ROOS debugging session with Igor, Paul and Don. We will attempt to use the ROOS discussion format to discuss the current state of ROOS, new features and ways to improve the user interface.

You have been invited to participate in a new discussion. Below, you will find the proposal that will form the basis of this discussion, as well as the format for the discussion and a list of legal additions to the discussion.

Subject: [test proposal1] proposal1
Date: Wed, 15 May 2002 09:41:28 -0700 (PDT)
From: ROOS <test@roos.com>
Reply-To: roos@mit.edu

To: hull@mit.edu
Please DO NOT modify the subject line of the message in any way, unless you are modifying a pre-existing attribute.

This discussion will end on 12:03:00 on 12–19–2001

Here’s a list of legal responses to this message.

<table>
<thead>
<tr>
<th>Action Tag</th>
<th>Description</th>
<th>Special Attribute (if Any)</th>
<th>Attribute Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>issue</td>
<td>A problem or obstacle that might hamper the discussion</td>
<td>duration</td>
<td>10</td>
</tr>
<tr>
<td>actionItem</td>
<td>A task that needs to be performed by an individual in order to proceed</td>
<td>duration</td>
<td>10</td>
</tr>
<tr>
<td>callToVote</td>
<td>A call to vote on an issue, proposal or discussion item. Must include a “second” before the vote proceeds. Only one such call is allowed, per subsection of a discussion.</td>
<td>duration</td>
<td>10</td>
</tr>
<tr>
<td>discourse</td>
<td>The general discussion component included in issues, proposals, and counter proposals.</td>
<td>duration</td>
<td>10</td>
</tr>
<tr>
<td>data</td>
<td>A link to any supporting data for the proposal or counter proposal that includes this contribution.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The following users have been invited to participate:

User Name
one such call is allowed per subsection of a discussion.

The general discussion component included in issues, proposals, and counter proposals.

A link to any supporting data for the proposal or counter proposal that includes this contribution.

The following users have been invited to participate:

<table>
<thead>
<tr>
<th>User Name</th>
<th>Full Name</th>
<th>Email Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>Igor</td>
<td>Igor L Belakovski</td>
<td><a href="mailto:bull@mit.edu">bull@mit.edu</a></td>
</tr>
<tr>
<td>Paul</td>
<td>Paul Dick</td>
<td><a href="mailto:paul_dick@us.ibm.com">paul_dick@us.ibm.com</a></td>
</tr>
<tr>
<td>Don</td>
<td>Donald Leslie</td>
<td><a href="mailto:donald_leslie@us.ibm.com">donald_leslie@us.ibm.com</a></td>
</tr>
</tbody>
</table>
The ROOS Collaboration Initiation Page

Short Collaboration Name: 

Proposal: 

Select Discussion Template: technical proposal town hall meeting Other: 

Is this collaboration public? yes no 

Initiator information: User name for discussion: Email: Full Name (F M L): 

Participants. Use the >> arrows to select known users and add them to the address book. Use << to remove users, and add new users by filling out the new user fields and clicking add to list. (Note that the initiator is automatically invited.)

Address Book:
Igor
David_Marston
Don
Emily
Paul
Joe
Shane
Merris

Invites:

Add new user:

User name: Email: Full Name (F M L): Add to list: 

[Signatures and timestamps]

Igor Belzakovsky
This discussion is set up to demonstrate the many wonderful features of ROOS.

Actions you may take:

- **issue**
  A problem or obstacle that might hamper the discussion.

- **actionItem**
  A task that needs to be performed by an individual in order to proceed.

- **callToVote**
  A call to vote on an issue, proposal or discussion item. Must include a "second" before the vote proceeds. Only one such call is allowed, per subsection of a discussion.

- **data**
  A link to any supporting data for the proposal or counter proposal that includes this contribution.

Participants

- Igor L Belakovskiy  Total Messages:19
  accepted

- paul_dick@us.ibm.com  Total Messages:8
  accepted

Messages

**proposal (1): Igor L Belakovskiy**

Now testing the web based attachment system. this attachment should be available at http://xsl-redhat.lotus.com/ROOS/attachments/5/14/(See attached file: mehosr.bmp)

This section and its children will expire at 10:54:00 on 12-20-2001

**data (14): Igor L Belakovskiy**

Now testing the web based attachment system. this attachment should be available at http://xsl-redhat.lotus.com/ROOS/attachments/5/15/(See attached file: mehosr.bmp)

This section and its children will expire at 10:54:00 on 12-20-2001

**data (15): Igor L Belakovskiy**

Now testing the web based attachment system. this attachment should be available at http://xsl-redhat.lotus.com/ROOS/attachments/5/16/(See attached file: mehosr.bmp)

This section and its children will expire at 10:54:00 on 12-20-2001

**data (16): Igor L Belakovskiy**
New testing the web based attachment system. this attachment should be available at http://xsl-redhat.lotus.com/ROOS/attachments/5/17/(See attached file: mehosr.bmp) this should finally work, the previous three did not..

Attachments:
mehosr.bmp

This section and its children will expire at 10:54:00 on 12-20-2001

issue (12): Igor L Belakowsky
attachment text/See attached file: steve.jpg

Actions you may take:

- actionItem
  A task that needs to be performed by an individual in order to proceed.

This section and its children will expire at 10:54:00 on 12-20-2001

discourse (13): Igor L Belakowsky
Testing if the system works if no attachments are sent.

Actions you may take:

- position
  State your positional on the issue/proposal being discussed. Must “take sides”, i.e. be for or against the proposal. Not a vote.
- clarifyingQuestion
  Ask a clarifying question about this
- strawVote
  A non-binding poll type vote. Multiple straw votes can be held.
- counterProposal
  propose something substantially different from the original proposal
- comment
  A general discussion comment. Meant for informal conversation

This section and its children will expire at 10:55:00 on 12-20-2001

callToVote (19)
vote (21)
second (20)

This attachment system might need a little bit more work, but it's still pretty cool.

Actions you may take:

- reply

reply (22)
vote (20)
second (19)

position (2)
clarifyingQuestion (9)
Chapter 4

Architecture

The design of the entire ROOS system was centered around the discussion and discussion template representations. In the first two sections of this chapter I will discuss those representations. In the next section I will describe various applications that are being used by the system. Finally, the final few sections discuss the various modules of ROOS.

4.1 Discussion Template Representation

In order to represent a discussion template, we had to pick a representation that would allow us to easily encode a discussion. We chose XML Schema [10] a subset of extensible mark-up language (XML) [11] as our representation. XML Schema allows the user to define an abstract data model for an XML document. This includes defining individual element properties, as well as the relationships between different elements and the document itself, such as order of appearance, number of appearances per node, and other structural properties. A schema can be easily designed to include very few required elements, but allow for a great number of optional elements, permitting the creation of very flexible discussion templates.

Schema also allows for easily creating and extending element types, which could be used as building blocks for a discussion component. Thus, more complex elements can be created by inheriting from the basic element structure, making sure that all
elements contain a base set of sub-elements and attributes. In this particular implementation, all elements representing a discussion contribution must be either of the componentType data type in figure 4-1, or extensions of that data type. The data type requires that a basic contribution has a content node with the text of the contribution; and a set of attributes, containing contributor information, a contribution identification number, a parent message and a boolean specifying if there were any attachments to the message. More advanced contribution types, such as a vote (see section 3.3.1), contain other attributes, as well. In the case of a vote contribution definition, the extra attribute is the actual vote cast by the user.

<complexType name="componentType">
  <sequence>
    <element name="content" type="string"/>
  </sequence>
  <attribute name="contributor" type="string" use="required"/>
  <attribute name="contributionID" type="int" use="required"/>
  <attribute name="parent" type="int" use="required"/>
  <attribute name="attachments" type="boolean"/>
</complexType>

Figure 4-1: Discussion Component Definition

Discussion structure within the schema is defined by extending the basic componentType to include other elements which also extend it. For example, the discourseType definition states that it may contain an element that conforms to the clarifyingQuestionType definition\(^1\). The clarifyingQuestionType definition in turn extends the basic componentType to state that elements conforming to answerType and actionItemType may appear. The actionItemType definition does not allow extra elements beyond the basic definition, and merely contains extra deadline attributes. However, the answerType definition states that a clarifyingQuestionType element may appear in it. This set of definitions is equivalent to the context free grammar in Figure 4-3 and any branches of discourse in figure 4-4 would be permitted in a discussion.

\(^1\)discourseType allows for other elements to appear, as well. To make this example easier to follow, those elements are ignored.
The power of XML Schema lies in the fact that it allows the user to define context free grammars (CFGs)\cite{21} to define the set of all possible discussion sequences, and to subsequently test whether or not appending or inserting a particular component will keep the resulting sequence in the set of all sequences defined in the original CFG. Schema also allows a computer application to verify whether or not a particular dis-

\footnote{Note that these are only \textit{SOME}, not \textit{ALL} of the legal sequences.}
• discourse → content | content · clarifyingQuestion

• clarifyingQuestion → content | content · actionItem | content · answer | content · actionItem · answer

• answer → content | content · clarifyingQuestion

• actionItem → content

Figure 4-3: Grammar Equivalent of Schema Definitions.

discussion conforms to the pre-defined template. Thus, schema solves both the problem of encoding a template, and that of enforcing it on a discussion.

(1) discourse
   \ 
   clarifyingQuestion
   \ 
   actionItem

(2) discourse
   \ 
   clarifyingQuestion

(3) discourse
   \ 
   answer
   \ 
   clarifyingQuestion

Figure 4-4: Legal Instances of Figure 4-2 Type Definitions.

4.2 Discussion Representation

Once XML Schema was chosen as the template language, the actual discussion representation followed naturally. Since template based moderation should be as fast as possible, the best way to represent and store a discussion would be as an XML document. XML or Extensible Markup Language is similar to the Hypertext Markup Language (HTML) in that both languages use meta tags to enclose data. However, HTML has a predefined set of meta tags, while XML tags are user defined, making
XML a much more flexible language than its older counterpart, HTML\(^3\) The tree-like nature of XML is a natural representation for e-mail threads, which are essentially trees with zero or more branches at each node. XML thus solves the problem of storing both discussion state and discussion content using a single representation. There was no need to create a separate discussion data type for maintaining the state of the discussion, a DOM [9] object was sufficient. Finally, storing the discussion as an XML document facilitates the transformation of the discussion using extensible stylesheet language (XSL) [12] for presentation and processing purposes(See section 4.4.3 for more details).

A minimal valid document conforming to design discussion schema can be found in Figure 4-5\(^4\). The conforming document contains schema and name space information, discussion id number, active status, title, availability, and the start time in the parent node. The discussion\(Statement\) node simply states the goals of the discussion as defined by the user in the discussion proposal (see Section 3.2). The participants node contains a child node, invitee for every discussion participant, containing information such as name, email address, user name, the acceptance status of the user, and the total number of messages sent by the user. In the example in Figure 4-5, user Igor has sent one message by initiating the discussion, and has thus changed his invitation status to accepted. His full name and e-mail address are attributes of the corresponding invitee element. Scott, the other participant in the discussion has not responded, and thus his participation status is still “pending” his response. Finally, the body node of the document is the parent node of the proposal node, which in turn is the root of the actual discussion thread. In this case, there is only one discussion contribution, proposal which is an extension of the basic componentType. In this case, the extension is just two additional attributes which specifies the starting time and duration of this element. Since this element also happens to be the root element of the discussion tree, these two attributes form the deadline for the entire discussion,

\(^3\)It should be noted that a well-formed (all tags are terminated properly) HTML document is also a legal XML document.
\(^4\)the corresponding Schema can be found in Appendix A

47
and not just a subsection of it.

<?xml version="1.0" encoding="UTF-8"?>
<disc:discussion activated="true" discID="9" discTitle="test proposal"
isPublic="true" startTime="2001-12-18T12:04:00"
totalContributions="1"
xmlns:disc="http://xsl-redhat.lotus.com/DISC"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://xsl-redhat.lotus.com/ROOS/new-discussion.xsd">
<discStatement>testing schema re-location</discStatement>
<participants>
<invitee email="ilbelako@us.ibm.com"
 fullName="Igor L Belakovskiy" status="accepted"
totalMsg="1">Igor</invitee>
<invitee email="Scott_Boag@us.ibm.com"
 fullName="Scott Boag" status="pending"
totalMsg="0">Scott</invitee>
</participants>
<body>
<proposal contributionID="1"
 contributor="ilbelako@us.ibm.com" duration="1"
 parent="1" startTime="2001-12-18T12:04:00">
<content>testing schema re-location</content>
</proposal>
</body>
</disc:discussion>

Figure 4-5: Minimal Discussion Document

4.2.1 Repository

Each ROOS discussion is stored in the discussion repository shared by all the ROOS modules. The configuration module only accesses the repository to create the initial discussion, the discussion module both reads and modifies existing discussions and the presentation module has read-only access to the repository to provide a Web portal to the discussion. Each discussion is stored in as a separate XML document. Additional documents are generated to assist the presentation module, but are not actually required for persistent storage of the discussions themselves. (See section
5.1.2). The discussion repository itself is a part of the file system where the XML documents described above are stored. In the current implementation of ROOS, the repository is located on the same physical server as the other ROOS modules. Using a file system such as the Andrew File System (AFS) would allow the repository to be seamlessly relocated to a separate server, in order to reduce the load on the server machine.

4.3 Tools

This section provides a brief overview of the tools used by the system to perform various tasks. The module diagrams in Figures 4-8, 4-9, and 4-10 will contain the interactions of the tools with each component of the system. Please refer to this section as well as the module diagrams for a complete understanding of the functionality and relevance of each tool.

All XML documents used by the system are accessed through Xerces, the Apache Software Foundation’s (ASF) XML parser. In particular, the Xerces implementation of the World Wide Web Consortium’s (W3C) Document Object Model[9] is used to parse, update and store various XML documents, including discussion documents, as well as auxiliary presentation files (see Sections 4.4.2 and 4.4.3). All accesses of Xerces were performed through the Java API for XML Processing (JAXP) [16].

XML Schema discussion template files are accessed using the schema processing functionality of the Web Sphere Application Developer (WSAD) created by the IBM Software Group in Toronto, Canada. WSAD is used by the configuration and discussion modules, for discussion initialization and discussion moderation, respectively. For performance and technical reasons, WSAD is not used by the presentation module. More on this can be found in Section 5.1.

All e-mail traffic is handled by the Java Apache Mail Enterprise Server (JAMES), another ASF project. JAMES is a pure Java e-mail server that supports the POP3, SMTP and IMAP email protocols.
All XSL transformations are performed by Xalan, the ASF’s XSL Transformation Engine. Xalan can transform any XML document into any other format by using XSL stylesheets.

All servlets, including the Xalan Transformation Servlet, and the configuration servlet are run by Tomcat, the ASF’s servlet engine. Tomcat can be easily configured to run under Apache Web Server, the industry standard web server, which is also used here, to serve all HTML content to the users. The web server and Tomcat are both used for the configuration and the presentation modules. Finally, the entire ROOS system ran under Red Hat Linux 7.1, on an IBM NetVista personal computer, with an Intel Coppermine 866MHz processor, 256MB of RAM, and a 30GB IDE hard disk drive.

4.4 Modules

The ROOS system can be broken into three functional components— the initialization/configuration portal, the discussion/moderation engine and the presentation server. Each of these components was designed as a separate stand-alone module. The decreased complexity of the system results in a much more scalable and testable design. The modularity of the system also permits for future improvements to be made on a module-by-module basis, as opposed to a system-wide upgrade\(^5\). The three modules also share the XML repository described in section 4.2.1 for discussion data storage and retrieval.

4.4.1 Configuration Module

The configuration module is responsible for discussion initiation. The module consists of a user interface for discussion initiation, the configuration servlet, and various interfaces to other modules and the common XML discussion repository. The user interface is an HTML form, with embedded JavaScript (see figure 3-7). The configu-

\(^5\)This will be discussed further in section 6.3
ration servlet is written in Java, and implements the Java Servlet API[5]. The actual
discussion configuration, however, is not performed by either one of these compo-
nents. Rather, the configuration of the discussion is embedded in the user-selected
(and possibly designed, see section 3.2) XML Schema template of the discussion.

The servlet uses WSAD in library mode to generate minimal discussion docu-
ments. WSAD returns DOM objects, which are then used by Xerces to generate
the XML discussion file. Once the document is generated and stored on the file sys-
tem, the configuration servlet never accesses that document again, and notifies the
discussion module that a new discussion has been created.

The notification of the discussion module occurs via email. The configuration
servlet use the JavaMail API[13] to generate and send electronic mail messages. Since
JAMES was installed on the same physical machine as all the modules, it is responsible
for both sending and receiving the notification message.

4.4.2 Discussion Module

The discussion module is responsible for the majority of the functionality of ROOS.
It is responsible for receiving messages, using the discussion templates and deadlines
to determine if those messages are legal, and forwarding them to discussion members
if they are, as well as updating the discussion XML file, for persistent storage and
presentation.

The discussion module consists of an e-mail account, which is used to send and
receive all e-mail messages associated with ROOS, and a discussion moderator appli-
cation. The moderator uses various APIs to communicate with James, WSAD and
Xerces to accomplish its various tasks (see Figure 4-9).

Email Account and Messages

The e-mail account for ROOS was set up using James, permitting access to it through
POP3 and SMTP protocols. The moderator application used JavaMail to retrieve
messages from the account, as well as to send messages to users. Since James was the
only mail server running on the system, it was responsible for communicating with JavaMail, and sending and or receiving the appropriate messages.

**Message Processing**

The discussion moderator's basic functionality is very straightforward—the moderator logs into the e-mail account, retrieves all messages received since last login, and processes each message. Based on the message type, the moderator sends out a message. In general, unless an unforeseen power outage occurs, for every message that the moderator receives, it sends out a message. In addition, several other message types are sent out, to facilitate discussion flow.

The discussion moderator receives two basic message types—discussion contributions from users, and discussion initiation requests from the configuration server. The former is a fairly complex task, which will be discussed in the next few paragraphs. The latter merely involves retrieving participant and proposal information from the XML discussion document created by the configuration servlet, and retrieving discussion structure information from the discussion schema. Once this information is retrieved, a message is sent to all participants, containing the text of the discussion proposal, as well as the first set of legal follow up contributions.

Legal follow up contributions for a particular message are determined from the schema definition of the message type. For example, messages conforming to the *clarifyingQuestionType* definition in Figure 4-2, would have two legal follow up messages—*actionItem*, and *answer*. Note that it is the element’s name in the type definition that is actually responsible for the tag of the contribution in the instance document, not its type. Thus, legal contributions are *answer* and *actionItem*, and not *answerType* and *actionItemType*.

The schema is also responsible for determining whether or not a particular message from a user is a valid contribution. The schema type definition of the parent message of the contribution (the message being replied to) is consulted. It is important to note that the particular message being replied to matters, not merely its type. This is due to the fact that some contribution types allow for only one instance of a follow
up message type to be added. For example, in any type of motion, only one other message is needed to second it. Therefore, if a message to second a motion arrives, it may be legal to add one instance of it to the discussion document, but not two. In order to determine this, the schema definition, as well as the instance document are examined. If a “second” message has already been posted, all subsequent attempts are rejected. If not, the first message to second a motion is accepted, and all subsequent ones rejected as superfluous. From a practical standpoint, this reduces clutter in users’ mailboxes. This is true for any message type that has limits on the number of contributions of a particular type.

In general, for a message to be accepted, the user must have chosen an element name for his contribution that is listed in the schema definition of the parent element of the contribution. In most cases, this involves the user clicking on mailto links in the message (see section 3.3), which are generated directly from the schema type definition. If the user changes the message type of the reply after selecting the link, by changing the subject line of the message, he may well find that his message has been rejected by the system, for having an illegal tag. Other built-in validity controls limit users to one vote per issue, with no option of reversing a vote once it has been cast.

However, before the structural validity of a particular message is examined, its timeliness is considered. Thus, a legal contribution to a discussion item that expired will not be tested for legality, but instead will be rejected as late. This greatly improves the efficiency of the system, as legality testing is the most resource consuming part of the moderation process. The expiration date on a contribution is determined either from the attribute information of the immediate parent of the element, or from the nearest ancestor that contains attribute information. For example, while neither a clarifying question nor an answer type contain expiration information, the discourse element does. Therefore, to determine the timeliness of a contribution tagged as an answer, the deadline-related attributes in the discourse element will be used, even though the immediate parent is a clarifying question element.

Once the structural and temporal legality of a contribution is ascertained, the moderator modifies the instance document corresponding to the discussion, by in-
serting an element in the discussion tree as a child of the parent contribution. A new contribution number is generated, and the new contribution becomes officially legal. At this point the new message is sent out to all the discussion participants, with updated discussion state information (see Section 3.3 for more details on what the user actually receives).

**Additional Functionality**

In addition to simply moderating user messages, the discussion moderator is also responsible for sending out various administrative messages. These include deadlines announcements, reminders, and vote summaries. In order for these messages to be created, the moderator maintains an event log. For each new contribution that has a deadline, a new set of events is added to the event log. For events over a day long, three events are generated—a halfway reminder, a 12 hours to go reminder, and the deadline notification. New events are generated and added to the log as soon as a contribution is accepted. If the moderator is restarted, events are generated from the discussion XML file, using the `startTime` and `duration` attributes. Since a vote tally occurs at the conclusion of a vote, a vote deadline announcement is directly followed by a vote count. Beyond message processing and generation, the discussion moderator also generates an up-to-date legal actions file for the discussion, whenever an element is added. While this may seem as a resource drain, in reality, when an element is added, only the element and its ancestors are affected by the change, and therefore the processing power consumed is minimal. The legal actions file is not actually used by the discussion server; instead, it serves as a source of data for the presentation server. Due to several problems with WSAD described in section 5.1, attempting to generate legal transition tables from the web in real time was simply not feasible. Furthermore, the legal actions are not required to be generated in real time, but merely after the last modification to the discussion. A simple locking scheme ensures that the presentation server does not attempt to read the legal action files while they are being modified by the discussion moderator.
**4.4.3 Presentation Module**

The presentation module is the simplest part of the ROOS system. The entire module is a servlet shipped with Xalan (see Section 4.3), which performs XSL transformations, a stylesheet for rendering various discussion components, the XML repository, and the legal actions files described above. The legal action files are XML documents, containing a node for every element in the discussion XML document, and a list of all legal tag names for each element as children of those nodes. A typical entry in a legal actions file can be found in figure 4-6. Note that special attributes are listed as well, since those too must be derived from the schema definition (see section 5.1.2). The basic attributes of *componentType* are common to all contributions, and therefore do not need to listed.

```
<component contributionID="1">
  <legalAddition>issue</legalAddition>
  <legalAddition duration="10">actionItem</legalAddition>
  <legalAddition duration="10">callToVote</legalAddition>
  <legalAddition>data</legalAddition>
</component>
```

Figure 4-6: Typical Entry In A Legal Actions File

**Stylesheet**

The discussion XSL stylesheet along with XSLT extensions is responsible for presenting discussions to users on the web. XSL is a language for expressing styleheets. It is used to define a template for transforming the date in an XML document from XML into another form, such as PDF, plain text or HTML. The use of style sheets allows users to separate style and presentation from data. The XSL stylesheet used by ROOS defines three HTML frames—a search frame, an index frame, and a content frame; and generates content for them (see Figure 3-8 for a screen shot of the stylesheet output). The top (search) frame allows the user to redisplay the discussion only with messages that contain a string that the user is searching for. This is a fairly
rudimentary search, and could be greatly improved (see Section 6.3.3.

The left frame is generated by simply traversing the xml document tree, and retrieving tag names and contribution numbers, as well as the expiration dates for each contribution. Using Java stylesheet extensions, the stylesheet renders active and expired messages differently, to aid visualization (in Figure 3-8, expired message branches are grey, and active message branches are black).

The right frame is generated from both the XML discussion document and the legal actions file. All author information, tag name, content, and expiration date is derived from the discussion document. The table of legal transitions is derived from the legal actions file and the tag description file. The tag description file provides descriptions of each each discussion component, and any special attributes that these components may possess. The file is also used by the moderator when creating the legal transitions table. A typical entry in the file can be found in Figure 4-7. The text field of the vote element is the description of the component itself, while the attr element contains a description of the special agreement indicator attribute of the component (isFor in the figure).

```
<vote>
   An individual vote. May include the user's comments about the way he/she voted
   <attr name="isFor"
       default="Yes">The value of the vote. Legal values are [Yes|No]. Set automatically through the mailTo: link
   </attr>
</vote>
```

Figure 4-7: Typical Entry In The Tag Descriptions File
Figure 4-8: The Configuration Module
Figure 4-9: The Discussion Module
Figure 4-10: The Presentation Module
Chapter 5

Issues and Alternative Implementations

During the implementation process, some changes were made to the architecture of ROOS to accommodate for performance and availability issues.

5.1 WSAD

Perhaps the biggest issues faced during the development process was the lack of a stand-alone schema processor. Since the XML Schema recommendation was only finalized in May of 2001, schema processors are only coming to market now, almost a year later. At the time, no such application was available. Instead, I used the WebSphere Application Developer (WSAD) code base, and a schema plug-in which was developed primarily for the Graphical User Interface of the WSAD editor. WSAD was developed by the IBM software group in Toronto, as interactive developer environment (IDE) for developing applications for IBM’s WebSphere product, as well as general development in Java and C. WSAD acts as an authoring tool for creating XML and HTML documents as well. It uses a plug-in mechanism to allow developers to add on to the basic environment, and create more advanced and sophisticated environments for development, as well as more powerful applications. WSAD can run in headless (command or batch) mode, however, all of its plug-ins had to be accessed
though the main runtime environment, and are not available separately, as a set of libraries. This was acceptable for the moderation server, which runs as a stand-alone application, and could simply be developed as a WSAD plug-in, and operate within the WSAD runtime environment. However, since both the configuration and the presentation modules had to run under the Tomcat Servlet Engine runtime environment, neither one of the above could have access to the standard WSAD plug-ins. In each case, a solution was found. Performance and procedural demands made it imperative that each case had a different solution.

5.1.1 Configuration.

The configuration server needs to be able to access the design schema created or chosen by the user during the configuration process. In order to retrieve schema information, the servlet had to use WSAD. However, to use WSAD, the servlet had to operate in the WSAD runtime environment, which could not be accomplished while still operating with a servlet engine. Instantiating a WSAD driver class within a servlet class, and using it to call the schema plug-in was the solution here. The WSAD driver class had to be modified, to allow for repeated calls to the same plug-in. Before the modification, certain calls would result in runtime errors. These errors were due to the fact that the driver classes were attempted to instantiate classes that could only be instantiated once per instance of a Java virtual machine. Adding some flags to the driver classes allowed the driver to differentiate between the first, and all subsequent calls, which fixed the problem.

5.1.2 Presentation

Unfortunately, while the presentation module was facing the same problem as the configuration module, it had the additional requirement of having to rapidly produce schema related data. While WSAD did perform the tasks required, it was, by far, the slowest part of the system. Since the WSAD code base was very large, optimizing it for performance was a task far too complex for one developer with a fairly cursory
knowledge of the internals of the system. Instead, I chose to use the moderator module to generate legal action files as new contributions were added. Since at most a branch of the discussion was being updated, even in the worst case, the performance of the moderator was as good as generating legal actions for each message on the fly. For any discussion with more than one branch, the moderator would spend less time on generating legal actions.

The legal action file also scales better to a large number of users, who instead of having to wait for a schema generator to parse an XML document and generate a legal transition tags, merely have to wait for Xerces to parse the XML actions file, and produce output with no further computations. The files only have to be updated once, when an action is added, as opposed to the schema processor having to generate the same data every time a new user tries to view the discussion on the web. Essentially, the imperfections of WSAD forced the design of a simple, but effective data caching system. It should be noted that the satisfactory resolution of the aforementioned issues with WSAD took over two months.

5.2 Alternatives.

If ROOS was originally developed for a more commercial enviroment, it could have been designed as a WebSphere[3] application. WebSphere is an IBM product, which combines many of the technologies used in the development of ROOS, including XML and XSLT. Integrating WSAD into a WebSphere application would have also been significantly more straightforward. As ROOS was designed for an open source audience, it was imperative that it used open source tools in it’s development, and therefore WebSphere was not chosen for this project. Doubtless, certain parts of the development process took significantly longer as a result. On the other hand, picking and chosing open source components for the project, allowed me, for the most part\(^1\) to use a minimal set of tools, which minimized the time spent on learning how to use the tools, and reduced the footprint of the application itself. Overall, I feel that

\(^1\)with the notable exception of the schema processor
the right decision was made, however the WebSphere approach may warrant further investigation in the future.
Chapter 6

Results and Future Work

6.1 Results

ROOS was deployed for a test run at the Advanced Technologies Group at IBM research. The success of the system was limited. Most users found that the system was fairly easy to use, and did produce easily viewable archives. However, most users had trouble with the concept that the subject line was automatically generated, and that the appropriate way to respond to a message was to follow a mail-to link, and not hit the reply button in their email browser. A great majority attempted the latter, and bombarded the author with questions about why their messages were being rejected. This led to the development of a more intelligent rejection system in the discussion moderator, which was able to detect several of the most common errors, and send reasonable error messages back to the users. Many users also complained about not being able to understand what each transition meant. This led to the development of the current version of the transition table, which contains fairly detailed explanations for each message type. Finally, the web interface constantly evolved, as most users had at least something to add to it, such as color coding contributions if they were “dead” or “live”, and color coding vote contributions.

In general, it would seem that the web provided a richer user experience than the email client. This was largely due to the fact that the web interface was thread aware, and allowed users to easily navigate between messages in a particular discussion, and
visualize sub-threads of the discussion. Hopefully, while the web interface continues to improve, the pioneering work at IBM research in the field of email clients will result in an improved mail client user experience as well. In the future, as more users migrate to browser-based mail clients, both technologies could be utilized without a detriment to the user community. Finally, it should be noted that the testing period was fairly limited in length, and thus resulted in a limited amount of user feedback being incorporated into the product.

6.2 Conclusions

In the short term, ROOS proved to be an exciting idea with great potential. Most users have found it to be helpful in some way—either as an improved way to communicate, or at the very least, as an improved way to archive discussion content. In order for ROOS to become a full featured product, several areas of the product need to be improved. Some specific improvements are discussed in section 6.3. In general, the biggest area of improvement should be the discussion template library. Input from experts in the field of human-computer interaction (HCI) would provide users with discussion structures more appropriate for daily use. The sample schema provided in Appendix A is a fairly elaborate attempt at creating such a template, and was found to be usable by most participants. At the same time, however, many participants found the template wanting as discussion complexity and length increased.

I believe that the ideas behind ROOS could be applied to the real world, and hope that the prototype described here can serve as a springboard for future development of structured email discussions. Certainly, the area of electronic collaboration can use improvements in both efficiency and archival, and I hope that ROOS can contribute to both of those areas.
6.3 Future Modifications

ROOS could be improved in numerous ways—nearly every ROOS user had a suggestion as to how to improve the user experience. However, at least two items figured prominently on the user wish list—a better template creation mechanism, and a better search engine.

6.3.1 New Schema Processor

Before any new improvements are made, one component of the system has to be replaced with a more robust and better performing one. The WSAD provides a great UI, and most of the functionality required by this project. However, the performance of the system was severely degraded because WSAD was simply not designed as a stand-alone schema processor. As XML Schema becomes a more mature standard, XML parsers such as Xerces, are beginning to support a more complete set of schema processing features. Using this open-source parser for schema processing in addition to XML document parsing would make the entire project depend only on open source components, making future development possible in an open source environment, for which this product was designed for.

6.3.2 Online Template Editor

Currently the only way for a user to create a discussion schema is to use a schema editor to write their own. In the future, it would be nice to create a web portal which would allow users to generate their own templates without having to write actual schema code. However, in order for this process to be feasible, a more lightweight schema editor has to be integrated into the system, to make real-time schema validation possible. Going forward, however, a better template creation tool is absolutely essential, if this project is to become widely used.
6.3.3 Search Engine

Currently, the presentation server allows the user to search for a particular string within the discussion database. Clearly, for a very large discussion, a more complex search would be required. In order to perform such tasks, the discussion documents will have to be indexed, and a more powerful search engine installed to operate in conjunction with the presentation module. There are commercial search engines available from vendors like AltaVista, as well as free ones, from vendors such as FreeFind.Com. Depending on the scale of the project, one of the above solutions would be a welcome addition to ROOS. A highly indexed, and easily searchable archive with tagged discussion branches could potentially be very successful.
Appendix A

Sample Discussion Schema

<?xml version="1.0" encoding="UTF-8"?>
<schema targetNamespace="http://xsl-redhat.lotus.com/DISC"
   xmlns="http://www.w3.org/2001/XMLSchema"
   xmlns:disc="http://xsl-redhat.lotus.com/DISC">
  <annotation>
    <documentation xml:lang="en">
      Discussion schema for ROOS
    </documentation>
  </annotation>

  <element name="discussion" type="disc:discussionType"/>
  <complexType name="discussionType">
    <sequence>
      <element name="discStatement" type="string"/>
      <element name="participants" type="disc:participantsType"/>
      <element name="body" type="disc:bodyType"/>
    </sequence>
    <attribute name="discID" type="int" use="required"/>
    <attribute name="totalContributions" type="int" use="required"/>
  </complexType>
</schema>
<attribute name="discTitle" type="string" use="required"/>
<attribute name="startTime" type="dateTime" use="required"/>
<attribute name="isPublic" type="string" use="required"/>
<attribute name="activated" type="boolean" use="required"/>
</complexType>

<complexType name="participantsType">
    <sequence>
        <element name="invitee"
            minOccurs="1" maxOccurs="unbounded"
            type="disc:userType"/>
    </sequence>
</complexType>

<complexType name="bodyType">
    <sequence>
        <element name="proposal" type="disc:proposalType"
            minOccurs="1" maxOccurs="1"/>
    </sequence>
</complexType>

<complexType name="proposalType">
    <complexContent>
        <extension base="disc:componentType">
            <sequence>
                <element name="issue" minOccurs="0" maxOccurs="unbounded" type="disc:componentType"/>
                <element name="callToVote" minOccurs="0" maxOccurs="1" type="disc:callToVoteType"/>
                <element name="issue" minOccurs="0" maxOccurs="0"/>
            </sequence>
        </extension>
    </complexContent>
</complexType>
<complexType name="issueType">
  <complexContent>
    <extension base="disc:componentType">
      <sequence>
        <element name="discourse"
          minOccurs="0"
          maxOccurs="1"
          type="disc:discourseType"/>
        <element name="callToVote"
          minOccurs="0"
          maxOccurs="1"
          type="disc:callToVoteType"/>
        <element name="actionItem"
          minOccurs="0"
          maxOccurs="unbounded"
          type="disc:actionItemType"/>
      </sequence>
    </extension>
  </complexContent>
</complexType>
<complexType name="actionItemType">
  <complexContent>
    <extension base="disc:componentType">
      <attribute name="startTime" type="dateTime" use="required"/>
      <attribute name="duration" type="decimal" use="required"/>
    </extension>
  </complexContent>
</complexType>

<complexType name="discourseType">
  <complexContent>
    <extension base="disc:componentType">
      <sequence>
        <element name="clarifyingQuestion" minOccurs="0" maxOccurs="unbounded" type="disc:clarifyingQuestionType"/>
        <element name="strawVote" minOccurs="0" maxOccurs="unbounded" type="disc:callToVoteType"/>
        <element name="position" minOccurs="0" maxOccurs=""/>
<element name="comment"
    minOccurs="0"
    maxOccurs="unbounded"
    type="disc:commentType"/>
<element name="counterProposal"
    minOccurs="0"
    maxOccurs="unbounded"
    type="disc:counterProposalType"/>
</sequence>
<attribute name="startTime" type="dateTime"
    use="required"/>
<attribute name="duration" type="decimal"
    use="required"/>
</extension>
</complexContent>
</complexType>

<complexType name="counterProposalType">
    <complexContent>
        <extension base="disc:componentType">
            <sequence>
                <element name="data" minOccurs="0"
                    maxOccurs="unbounded"
                    type="disc:componentType"/>
                <element name="discourse" minOccurs="0"
                    maxOccurs="1" type="disc:discourseType"/>
            </sequence>
        </extension>
    </complexContent>
</complexType>
<complexType name="clarifyingQuestionType">
  <complexContent>
    <extension base="disc:componentType">
      <sequence>
        <element name="answer"
          minOccurs="0"
          maxOccurs="unbounded"
          type="disc:answerType"/>
        <element name="actionItem"
          minOccurs="0"
          maxOccurs="unbounded"
          type="disc:actionItemType"/>
      </sequence>
    </extension>
  </complexContent>
</complexType>

<complexType name="answerType">
  <complexContent>
    <extension base="disc:componentType">
      <sequence>
        <element name="clarifyingQuestion"
          minOccurs="0"
          type="disc:clarifyingQuestionType"/>
      </sequence>
    </extension>
  </complexContent>
</complexType>
<complexType name="callToVoteType">
  <complexContent>
    <extension base="disc:componentType">
      <sequence minOccurs="0" maxOccurs="1">
        <element name="second" type="disc:componentType"/>
        <element name="vote" type="disc:voteType" minOccurs="0" maxOccurs="unbounded"/>
      </sequence>
      <attribute name="startTime" type="dateTime" use="required"/>
      <attribute name="duration" type="decimal" use="required"/>
    </extension>
  </complexContent>
</complexType>
<complexType name="positionType">
  <complexContent>
    <extension base="disc:componentType">
      <attribute name="isFor" type="string"
                 use="required"/>
    </extension>
  </complexContent>
</complexType>

<complexType name="commentType">
  <complexContent>
    <extension base="disc:componentType">
      <sequence>
        <element name="reply"
                 minOccurs="0"
                 maxOccurs="unbounded"
                 type="disc:replyType"/>
      </sequence>
    </extension>
  </complexContent>
</complexType>

<complexType name="replyType">
  <complexContent>
    <extension base="disc:componentType">
      <sequence>
      
      </sequence>
    </extension>
  </complexContent>
</complexType>
<element name="reply"
    minOccurs="0"
    maxOccurs="unbounded"
    type="disc:replyType"/>

</sequence>
</extension>
</complexContent>
</complexType>

<complexType name="componentType">
    <sequence>
        <element name="child" minOccurs="0" maxOccurs="unbounded"
            type="string"/>
        <element name="content" type="string"/>
    </sequence>
    <attribute name="contributor" type="string" use="required"/>
    <attribute name="contributionID" type="int" use="required"/>
    <attribute name="parent" type="int" use="required"/>
    <attribute name="attachments" type="string"/>
</complexType>

<complexType name="userType">
    <simpleContent>
        <extension base="string">
            <attribute name="fullName" type="string"
                use="required"/>
            <attribute name="email" type="string"
                use="required"/>
            <attribute name="status" type="string"
                use="required"/>
        </extension>
    </simpleContent>
</complexType>
<attribute name="totalMsg" type="int"
    use="required"/>

</extension>
</simpleContent>
</complexType>
</schema>
Appendix B

Appache Traffic Analysis

<table>
<thead>
<tr>
<th>Day</th>
<th>Messages</th>
<th>Threads</th>
<th>Avg Length</th>
<th>Max Length</th>
<th>Min Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>30</td>
<td>17</td>
<td>1.76</td>
<td>7</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>48</td>
<td>20</td>
<td>2.4</td>
<td>17</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>17</td>
<td>10</td>
<td>1.7</td>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>37</td>
<td>19</td>
<td>1.95</td>
<td>12</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>32</td>
<td>13</td>
<td>2.46</td>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>22</td>
<td>11</td>
<td>2</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>9</td>
<td>11</td>
<td>6</td>
<td>1.83</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>20</td>
<td>8</td>
<td>2.5</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>11</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>12</td>
<td>6</td>
<td>6</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>13</td>
<td>47</td>
<td>16</td>
<td>2.94</td>
<td>8</td>
<td>1</td>
</tr>
<tr>
<td>14</td>
<td>52</td>
<td>9</td>
<td>5.77</td>
<td>20</td>
<td>1</td>
</tr>
<tr>
<td>Totals:</td>
<td>23,214,286</td>
<td>9,857,1429</td>
<td>2,022,1429</td>
<td>6,571,4286</td>
<td>0,928,57143</td>
</tr>
</tbody>
</table>

Table B.1: Xalan Dev Traffic Analysis.

Based on the Xalan-Dev mailing list Internet archive[7] over two separate periods, each a week long, beginning on March 7, 2002, and March 20th, 2002, respectively. Days 3, 4, 11 and 12 were weekends, as can be seen from the unusually low traffic flow.
Bibliography


