IMPROVING MAILING LIST ARCHIVES THROUGH CONDENSATION

A THESIS SUBMITTED TO THE GRADUATE DIVISION OF THE
UNIVERSITY OF HAWAI‘I IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF SCIENCE

IN

INFORMATION AND COMPUTER SCIENCES

MAY 2000

By
Robert S. Brewer

Thesis Committee:

Philip M. Johnson, Chairperson
Wesley Peterson
Edoardo S. Biagioni

We certify that we have read this thesis and that, in our opinion, it is satisfactory in scope and quality as a thesis for the degree of Master of Science in Information and Computer Sciences.

THESIS COMMITTEE

_____

Chairperson

_____

_____

©Copyright 2000

by

Robert S. Brewer

To Yuka: thanks for
all the love, support, and patience
you have shown me while I worked
on this document, which was go-
ing to be finished in “a year at
most”. It is most deeply
appreciated.
♡

Acknowledgments

Like any research project, this one could not have been completed without the help of a number of people. I'd like to thank my parents for putting me through college and encouraging my academic ambitions. Your lifelong scholarship set an example for me.

My thanks go to Philip Johnson, for putting up with me and my smart-alecky ways. You have always supported my efforts, even when they were at best vaguely related to the group's research focus. You even supported my decision to temporarily leave the program (though I think you should have taken me up on my investment offer :)

I'd also like to thank the past and present members of CSDL in chronological order: Dadong Wan, Danu Tjahjono, Rosemary Sumajit, Cam Moore, Anne Disney, Jennifer Geis, Russell Tokuyama, Joe Dane, Tie Fang, Monir Hodges, and Mette Moffett. Being able to bounce ideas off of all of you was incredibly valuable to me.

Kirill Levchenko deserves mention as someone outside UHM who provided feedback about the research. It was appreciated (even if your first comment on it was "weren't you going to do something cool?" :)

I would like to thank my committee members Wesley Peterson and Edoardo Biagioni for taking the time to read and evaluate this thesis.

Tim Endres is the maintainer of the mailing list which was used in the case study of MCS. His willingness to let his list be used and his general support is appreciated.

Last but not least, I thank Yuka Nagashima for all that she has done in support of me. You have put up with griping, provided encouragement when I was depressed, and read drafts at the drop of a hat. Thank you.

Abstract

Searching the archives of electronic product support mailing lists often provides unsatisfactory results for users looking for quick solutions to their problems. Archives are inconvenient because they are too voluminous, lack efficient searching mechanisms, and retain the original thread structure which is not relevant to knowledge seekers.

I present MCS, a system which improves mailing list archives through *condensation*. Condensation involves omitting redundant or useless messages, and adding meta-level information to messages to improve searching. The condensation process is performed by a human assisted by an editing tool.

I describe the design and implementation of MCS, and compare it to related systems. I also present my experiences condensing a 1428 message mailing list archive to an archive containing only 177 messages (an 88% reduction). The condensation required only 1.5 minutes of editor effort per message. The condensed archive was adopted by the users of the mailing list.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 The Problem with Mailing List Archives

As the world economy shifts increasingly towards information, the computer hardware and software products we use and their interrelationships become more complicated. Users of the products often encounter problems and want to find solutions.

Electronic mailing lists provide an excellent way for users of a particular product to exchange information and help each other. While some mailing lists are created by the person or organization that created the product, others are started by interested users. Discussions on the lists include feature requests, bug reports, and reviews, but the most common topic is problem solving. Users who encounter problems send messages to the list documenting what happened and other users (possibly the vendor itself) respond with possible solutions.

Many mailing lists store the messages sent to the list in an archive for future retrieval. These archives can range from giant text files to databases with sophisticated World-Wide Web (also know as WWW or web) interfaces. The two most common types of archives are:

- browsable thread-based archives that allow a user to read messages in a format similar to the one in which they were sent to the list
- searchable archives that allow users to do full-text searches over all the articles in the archive.

The mailing list format has a few important benefits: the barriers to participation are low, the software required to participate is widely available, and it makes use of widely deployed server-support (many mail servers have mailing list distribution functionality). Unfortunately, these very benefits prevent the fullest use of the information in the mailing list data stream. Since anyone

can contribute to the list and the format is very conversation-like, the result is a data stream with a mediocre signal to noise ratio at best. While there is a lot of valuable knowledge available, one often has to slog through endless newbie questions, flamewars, and “Me Too”s.

The problem worsens when you take into account the archives of the list. When users consult a mailing list archive, they are often searching for a specific piece of information like a solution to a problem they are having. Unfortunately mailing list archives are poorly equipped to support this kind of query. All the irrelevant information that was sent to the list is immortalized in the archive, making it difficult to find useful information. Searchable archives also face the problem that any particular query may return an enormous number of hits. For example, a search for “OSPF” (Open Shortest Path First, a modern TCP/IP routing protocol) on the ascend-users mailing list archive [1] returns 650 hits, which is an artificially imposed maximum. In this case the hits are displayed in reverse chronological order, which isn’t necessarily desirable if you are looking for a particular OSPF problem.

Some mailing list communities generate a list of Frequently Asked Questions (FAQs) for the mailing list. Originally the purpose of a FAQ was to avoid the recurring situation where new subscribers to the list would ask questions that had been asked and answered many times before. By creating a list of these frequently asked questions, newbies’ questions could be answered by the FAQ. The FAQ format is also used as a convenient format for disseminating useful information on the subject matter, i.e., questions that might not be frequently asked but are useful to know. The big problem with FAQs is that they are primarily maintained by hand. This means that keeping an FAQ up to date is a labor-intensive process which frequently exhausts the volunteer maintainer. FAQs are generally intended to be documents (text, HTML, etc.) which can be distributed (posted to a newsgroup, downloaded from a web page). This “distributable” quality limits the size and organization of the document to something that can be read and understood by a human reader. It also means that the there is rarely any searching facility provided for the FAQ (as such a process requires interactivity). The combination of these two factors limits the amount and depth of information that can be presented. In other words, a FAQ by definition leaves out useful information if it isn’t used frequently enough to justify its inclusion to the space-limited FAQ. An additional failing of FAQs is that they are not designed to deal with questions whose answer frequently changes. The best way to work around a bug in a product may change from version to version, and eventually become irrelevant when the bug is fixed. Keeping the FAQ up to date puts heavy demands on the FAQ maintainer.

1.2 Condensation as a Solution

To solve the problems inherent in current mailing list archives of product support mailing lists, I propose a process called *condensation* whereby one can strip out all the extraneous, conversational aspects of the data stream, leaving only the interconnected pearls of wisdom. Condensation also involves the editing of the data stream for maximum utility and the addition of meta-level information to support more efficient searching. This condensation takes the voluminous data stream from the mailing list and extracts the useful information. As an analogy, newspapers provide a daily report on current events but are limited by short deadlines, a broad subscriber base, and other considerations. These considerations prevent them from analyzing which events are accurate or relevant over the long term. A story published one day might be amended or retracted the next, depending on how events unfold. However, a book describing world events will tend to have a longer deadline which permits more reflection and analysis: a hoax which might occupy weeks of headlines in a newspaper will probably be little more than a footnote in a book (unless the book is about newspaper hoaxes). The book can also have an index to enable readers to jump directly to the information they are interested in. It is this refinement of information that I refer to as condensation.

More specifically, creating a *condensed* archive from a data stream involves several steps:

- Each message is read to decide if it is relevant to the condensed archive. Since the goal of the archive is to solve problems, messages which describe neither problems nor solutions are dropped.
- Editors add a variety of meta-level information to the message. They assign a type, write a one-line summary, add keywords, and extract symptoms.
- Editors can remove, add, or change the body of the message in order to increase clarity or provide context.

The result of condensation is a much smaller archive that contains problems which are linked to their respective solutions (when solutions exist). This process destroys the conversational structure of the list messages, but users with problems are looking for solutions, not conversation. The reduction in size of the archive in itself makes searching more efficient because there is less chance of receiving irrelevant search results. The added meta-level information enables new kinds of searches like the symptom search (see Section 2.1.2) which are simply not possible with traditional archives. Condensation also solves the problems facing FAQs. Since condensation relies on a centralized archive, it does not face the same size and complexity constraints that an FAQ

does. Unlike FAQs, condensed archives are good at handling questions whose answer changes frequently, because the answer to a question is a query to the archive database. Each time the answer is requested, it can be constructed from the latest information stored into the condensed archive. Condensation requires less effort than maintaining an FAQ, because the messages which form the raw materials for condensation are written by someone other than the editor.

1.3 MCS: Condensation Realized

To demonstrate the improvements possible through condensation, I have constructed a new software system for condensing mailing list archives. I have named this system (for lack of imagination) the Mailinglist Condensation System or MCS [2]. MCS has two main parts: one which is dedicated to taking the raw material from the mailing list and condensing it, and another which stores the condensed messages and allows users to access them.

One way to perform the condensation would be to implement an AI system that reads the messages and then decides what information to keep, what to throw away, and what keywords to assign to each. To perform this task adequately, the system would need superb natural language processing capabilities and an in-depth knowledge of the mailing list's domain. Such a system is currently at or beyond the state of the art, and would at any rate require a substantial investment of resources to develop and maintain.

A practical alternative to an AI system is the employment of human editors for condensation, along with extensive tool support to lower editing overhead to an acceptable level. Humans are quite good at examining textual information and determining what is useful and what is not, while computers are good at queries across structured data [3]. Using human editors is also far more resource efficient because most mailing lists already have a set of 'gurus': subscribers who read all messages sent to the list and who are domain experts. Therefore, in MCS, humans do the editing using the MCS editing program which makes the process as efficient as possible. Only the editors need to use the editing subsystem; the user interface to the archive itself is separate and geared towards ease of use.

The storage and retrieval subsystem of MCS consists of a web server connected to a database system. As editors create condensed messages, they are placed into the database by the editing tool. For ease of use, end-users access the condensed archive with a web browser.

1.4 Two Example Searches

To show how condensation can create a more useful archive than traditional methods, I will consider an example query for a list that has two conventional archives and a FAQ to show some of the difficulties users face when attempting to find solutions to problems. Then I demonstrate how a condensed archive can solve a similar problem with ease.

1.4.1 Example 1: Finding Solutions in Traditional Archives

The first example involves compiling and installing perl 5.004_04 on BSD/OS. Perl is a interpreted language which is designed for text processing [4]. BSD/OS is an operating system from Berkeley Software Design Inc. (BSDI) which runs on Intel-based computers [5]. BSD/OS has a user-maintained mailing list called "bsdi-users" for discussion of all issues about the operating system. This example uses the actual information available from the "bsdi-users" mailing list.

Perl 5.004_04 comes with an automated configuration program which configures and compiles the software when told what operating system it is running on. Unfortunately, there is a problem with the configuration script. When this version of perl was released, BSDI was working on a new version of BSD/OS. At that time, the new version was assigned number "3.1". The configuration script for perl was set up so that certain changes would be made if perl was installed on a 3.1 system. This would allow the same perl distribution to work both on the existing 3.0 system and the new 3.1 system when it was released. Unfortunately, the version which was to be called 3.1 was delayed and renumbered "4.0". A minor revision of BSD/OS was released with the (now unused) version number "3.1". As a result of this version number mix up, installing perl 5.004_04 on a BSD/OS 3.1 system would fail in strange ways due to the failure of some of the configuration assumptions. User confusion over this was compounded by the existence of the special configuration information for BSD/OS 3.1 in perl: since perl had configuration information preset for 3.1, surely it couldn't be wrong?

This problem has three interesting attributes:

1. The solution to the problem is fairly simple: a single configuration script in the perl distribution needs to have all instances of "3.1" changed to "4.0" and some other minor changes. The changes can either be made automatically through the Unix `patch` command or be made by hand (once you understand the cause of the problem).

2. The problem has obvious symptoms which definitively indicate the problem: inability to install perl on BSD/OS 3.1 or specific error messages that occur when attempting to compile perl with unpatched configuration files.

3. The perl distribution is updated with bug fixes infrequently so this problem was encountered by many people over several months. This caused people to post messages to bsdi-users asking for help with the same problem repeatedly over several months.

There are three well-known information sources related to or derived from the bsdi-users mailing list: the BSD/OS FAQ, the Support Net archive, and the Nexial Systems archive. I attempted to find the solution to the problem in each information resource.

The BSD/OS FAQ

The BSD/OS FAQ [6] is maintained by a single person as a volunteer effort. There are 73 question/answer pairs in the FAQ, and it is not immediately clear what method was used to order them. The word "perl" does not appear in version 1.1.0 (dated 98/12/07 [sic]) and there is no discussion of our example problem. It is possible that the question was in the FAQ at one point and then later removed when the perl distribution was fixed, but this seems unlikely given that there are other question/answer pairs in the FAQ which are more than two years out of date. The fact that it is not in the FAQ is not surprising. The FAQ is maintained in someone's spare time, so many useful question/answer pairs will never make it in due to time constraints. The FAQ is also a manageable size at about 37 kilobytes, allowing it to be posted to the mailing list on occasion. If the FAQ were to contain information every installation problem for every package used with BSD/OS, it would probably become unmanageable.

The Support Net Archive

The Support Net archive contains all messages posted to the mailing list over the last 12 months [7]. The messages are accessible in two ways: each month's messages displayed in a thread format, or searched using the Excite for Web Servers search engine (EWS) [8]. An example of the thread format is shown in Figure 1.1. The thread format is useful when trying to follow the flow of a conversation, but it is ill-suited to finding a particular piece of information.

The EWS search engine used by Support Net indexes a collection of documents and provides a "concept-based architecture" for searching. It claims to analyze the document collection and

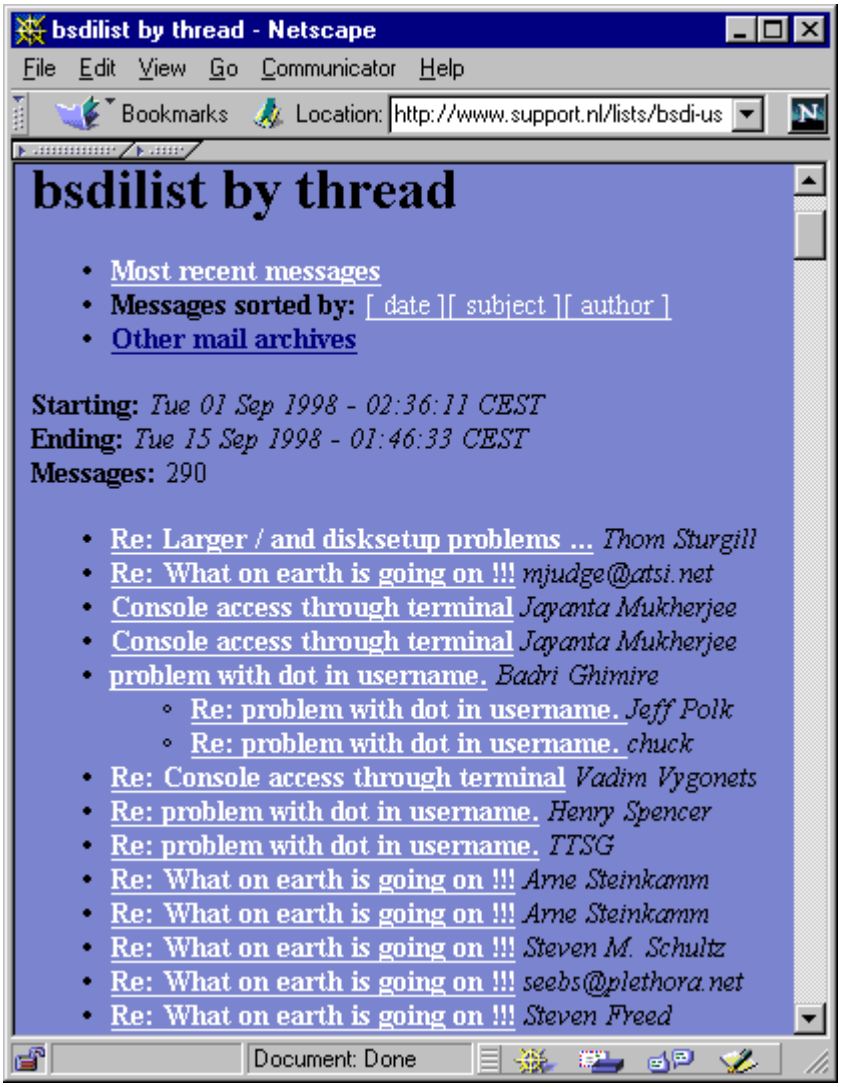

Figure 1.1. View of Support Net bsdi-users archive in threaded mode

determine “statistical correlations between terms and documents” which improves recall and precision compared to other search methods. In an attempt to find the answer to our example question, I performed a search with input “installing perl5 compile problems”. Part of the results can be seen in Figure 1.2.

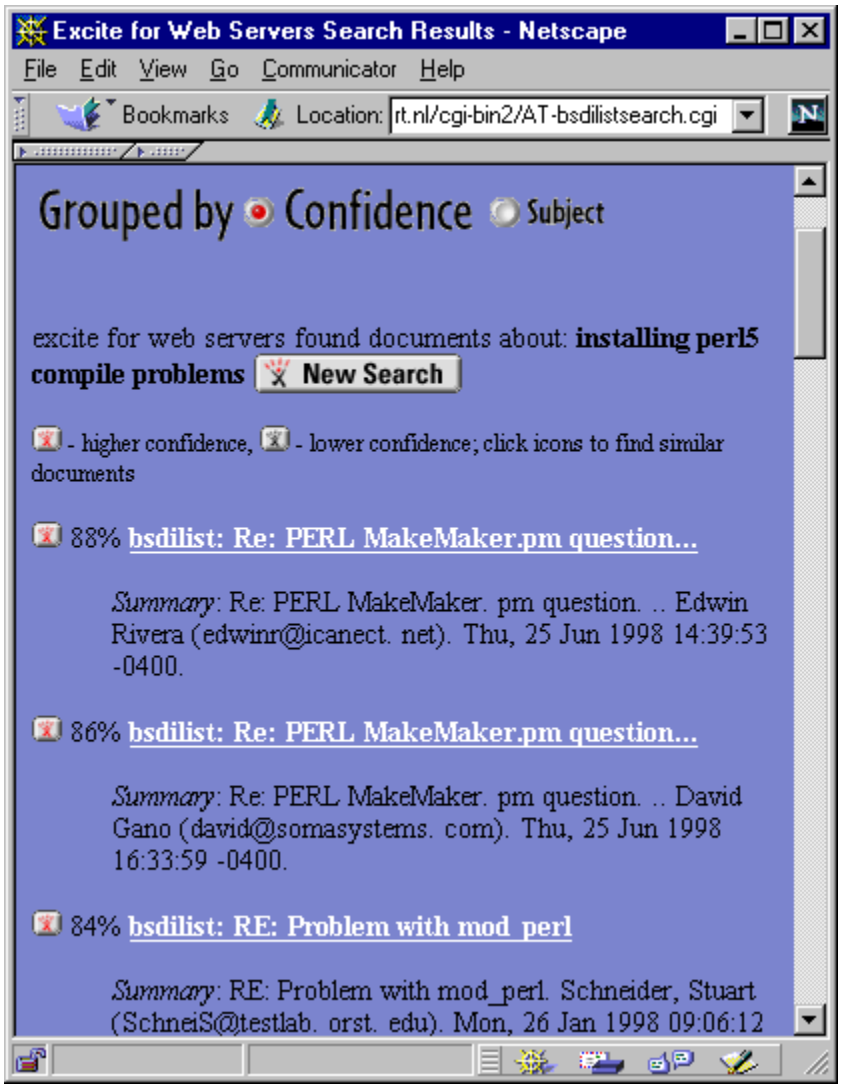

Figure 1.2. View of results from example search in Support Net bsdi-users archive

The initial search returned a few relevant messages, but they were all written by people who had encountered the problem and were asking for a solution. After examining the first set of messages, I used the EWS query-by-example feature to search for messages related to the most relevant of the search results. This search resulted in several more messages asking for help on the

problem, and a few misguided attempts to help. Again, I selected the message that best reflected the problem and performed a query-by-example. The next batch of messages included one which acknowledged the problem, but referred the person asking the question to the archives for the actual patch to solve the problem! After fifteen minutes and several more iterations of query-by-example, I obtained both a message containing the actual patch and a message from BSDI personnel explaining the problem's genesis (as previously summarized).

These results plainly show the problems with a traditional archive. Since every message is included in the archive, messages repeating the question are often returned by the search. Since the results are frequently sorted first by relevance and then by date, these repeat questions are actually more likely to be returned by a search than the first time the question was asked. The repeated posting of questions also reduces the likelihood that a useful response will be given, as evidenced by the "look it up in the archives" response and several repeats which did not appear to be answered at all. One message responding to a repeat asks that the question/answer pair be put in the FAQ, which we know has not been done! This shows that even though there was a recognized need for this problem to be documented in the FAQ, it never happened (for whatever reason).

The Nexial Systems Archive

The Nexial Systems archive provides a *fuzzy* search mechanism for the bsdi-users list [9]. The fuzziness allows the engine to find messages that match keywords that are spelled in a similar manner to the ones provided by the user. Starting with the same initial set of keywords as the search of the Support Net archives, I attempted to find the solution in the Nexial archives. Finding the answer in the Nexial archive took longer and required more effort because it does not provide a query-by-example facility. This required massaging the keywords until I found ones that matched the message containing the patch. Getting the keywords right also required me to extract keywords from some of the earlier search results, like the word "hint" which refers to the hint file used by the configuration system which the patch applies to.

The problems with traditional archives are the same in the Nexial archive. Most of the messages retrieved were users re-asking the question or answers which just say "consult the archives". This latter request is somewhat amusing considering that it is rather difficult to dig up the patch from either archive even when you know exactly what you are looking for. Another interesting point is that the cycle of re-asking the question and being referred to the archives is actually a feed-back loop. Each time someone asks this question, they increase the number of useless matches the

next person querying the archives will get. When someone cannot find the answer in the archives, the obvious alternative is to post the question to the list *again*.

1.4.2 Example 2: Finding a Solution in a Condensed Archive

For logistical reasons, the bsdi-users list was not the one which was condensed for the case study (see Section 4.1) of this research. The list that was condensed is the jCVS list [10] which is for the discussion of jCVS [11], a Java client for the Concurrent Versions System (CVS) [12]. My example of the condensation solution using MCS is therefore based on this list.

One of the enhanced searching techniques available in a condensed archive is the symptom search. This kind of search is designed to be easy and quick for users who have problems that generate diagnostic error messages. For example, say a user tries to run the jCVS tool for the first time and receives the following error message on their console:

```
java.lang.NoClassDefFoundError: javax/swing/DefaultBoundedRangeModel
        at com.ice.jcvsii.JCVS.instanceMain(JCVS.java:81)
        at com.ice.jcvsii.JCVS.main(JCVS.java:63)
```

Not knowing what the problem is, the user visits the condensed jCVS archive and switches to the symptom search mode, then pastes the above error message into the symptom field. The resulting screen is shown in Figure 1.3.

After clicking the Search button, the user immediately receives the search results shown in Figure 1.4. For this particular symptom, there is only one matching problem, and there is a matching solution. The summaries of both problem and solution are shown so the user can tell that this looks like a useful result. Selecting either problem or solution link displays the respective message.

The results here show the ease with which the desired information is found. The user was able to use the actual error message to initiate the search which is far more intuitive than trying to guess what keywords to use. Since the archive is condensed, the were only a few matches and they were immediately useful because the user can tell which message describes the problem and which describes the solution. MCS includes two other enhanced searching techniques made possible through condensation: symptom search, and 2D search. These search methods are discussed in Section 2.1.

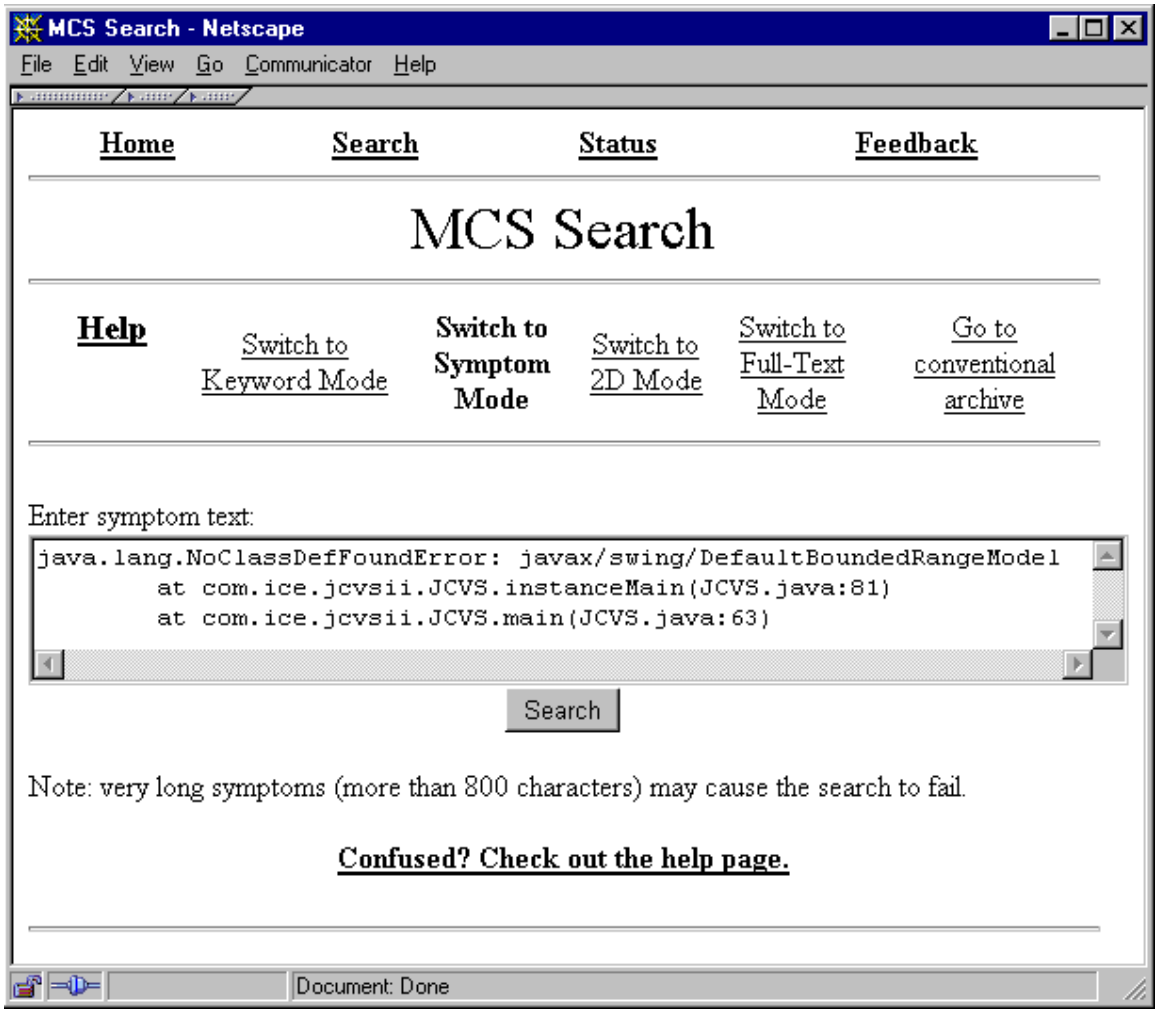

Figure 1.3. Example symptom search with initiated using an error message as input

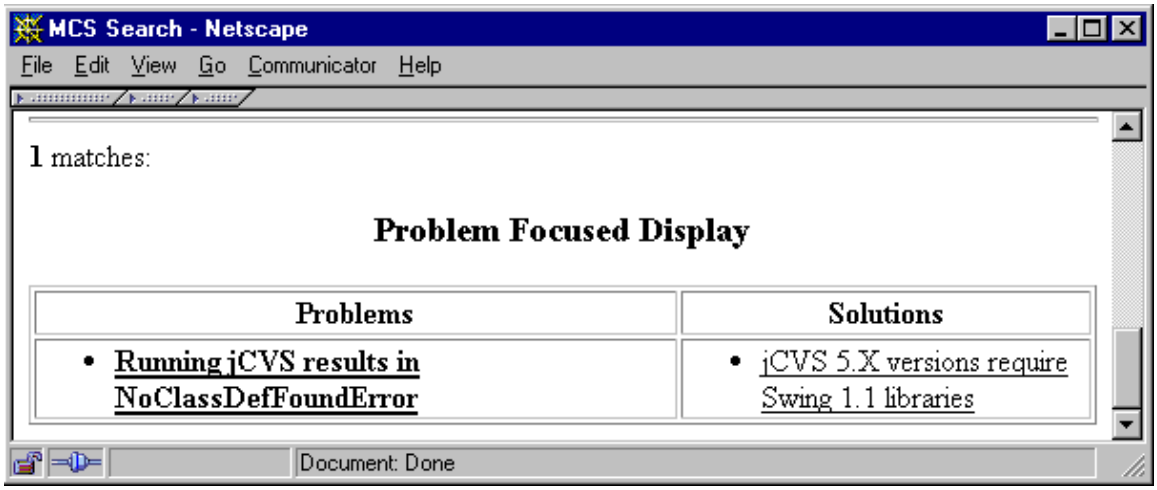

Figure 1.4. Results from example symptom search shown in Figure 1.3

1.5 Thesis Statement

This research has demonstrated three things:

1. Condensation of a non-trivial archive from a mailing list has been performed in a reasonable amount of time.
2. A mailing list archive condensed using MCS has been adopted by the subscribers of that list.
3. Subscribers preferred the MCS condensed archive to the existing, conventional archive of the list.

Statement 1 addresses the explicit trade-off that MCS makes by using the effort of a human editor to improve the archive for users. If condensation required substantial time per message (like 10 minutes), then editing a large archive would require an enormous amount of time. This would make use of MCS prohibitive except for those cases where some organization was willing to pay several editors to perform the condensation.

Statement 2 concerns whether subscribers will actually use the condensed archive. I define adoption as a significant fraction of the subscribers using the MCS archive either in addition to or instead of the traditional archives. I have used the number of list subscribers as an estimate of the number of potential MCS users. The adoption percentage is then the number of MCS archive users divided by the number of list subscribers, expressed as a percentage. To decide what adoption percentage would be indicative of success, I consulted Everett's work on the the diffusion of

innovations [13]. He divides adopters into five categories based on the rate at which they adopt innovations. The two categories containing the most rapid adopters are the *Innovators* (consisting of 2.5% of the population), and *Early Adopters* (consisting of 13.5% of the population). I decided to target both these categories, so my target adoption percentage is the sum of the category sizes: 16%.

Statement 3 addresses whether the archive users actually preferred the MCS archive to existing alternatives. The conventional archive is defined as the existing archive of a mailing list that allow either browsing of or searching through the messages posted to the list over time. For example, in Section 1.4 the traditional archives of the bsdi-users list are the Support Net and Nexial Systems archives. If users had not preferred the MCS-condensed archive then the additional manual effort required to maintain it might not be justified.

It should be noted that statements 2 and 3 assume that the archive is being maintained by myself as an external researcher. Unlike traditional archives, an MCS archive requires effort to maintain its usefulness. Therefore, future adoption of MCS by other mailing lists might fail despite the positive results presented here because there was no external agent performing the condensation "for free". The issue of editor recruitment is discussed in Section 5.2.3 and Section 7.2.

1.6 Overview of this Document

In the remainder of this document, I explain the MCS system in detail, and then describe a case study I designed to evaluate MCS. In Chapter 2, I describe the archive and editing interfaces of MCS. In Chapter 3, I explain the design and implementation of MCS. In Chapter 4, I present the case study I designed to evaluate MCS. In Chapter 5, I present the results I obtained through the case study. In Chapter 6, I compare MCS to related systems and techniques. In Chapter 7, I conclude with a summary of the research, and describe some possible future directions.

Chapter 2

Using MCS

MCS has two user interfaces: a web interface employed by users of the archive, and the mailer interface used by MCS editors to create and maintain the archive. This chapter will step through the functionality of both interfaces and point out how that functionality is obtained.

2.1 The Archive User Interface

Most archive users are subscribers of the mailing list that the archive is based on, although the archive allows anyone to use it regardless of their list membership status. To make it as easy as possible for users to access the archive, the user interface is purely web based. Users start their session by pointing their browser to the front page of the MCS-condensed archive. The front page of the condensed archive used in the case study (see Chapter 4) is shown in Figure 2.1.

From the front page the user can read more information about MCS or they can follow a link directly into one of the four search methods supported by MCS:

- *Keyword Search* allows the user to pick one or more keywords and see all messages which contain that keyword. Unlike most archives, the set of keywords is quite small and only contains words hand-selected to be highly relevant to this particular archive. The keywords are arranged into categories through which the user can browse before making his or her selection.

- *Symptom Search* is designed to help the user search the archive using an error message as input. If the user has encountered a problem which generates an error message, they can copy and paste it into the symptom field and the system will try to find any problems which exhibit that same symptom.

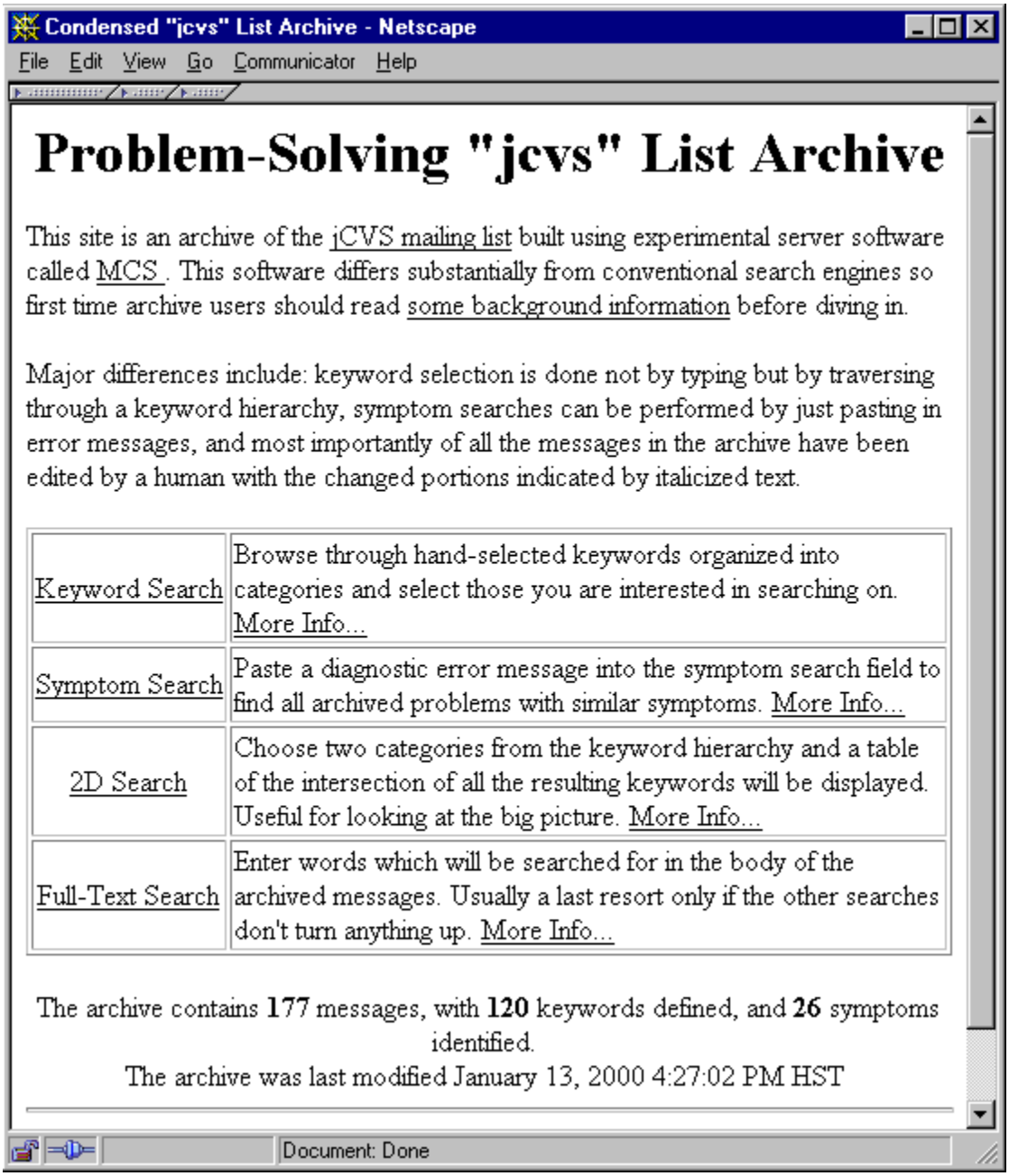

Figure 2.1. Initial page of an MCS-condensed archive

- *2D Search* allows the user to perform a unique kind of *intersection search*. As mentioned above, keywords are organized into categories for ease of browsing. To initiate a 2D search the user selects two different categories from the keyword hierarchy. MCS then creates a table that shows the results of searching for each keyword from the first category with each keyword from the second category.

- *Full-Text Search* allows searching based on words in the bodies of archived messages. This is the kind of search most people are familiar with, but this method is de-emphasized because this search does not exploit the benefits of condensation to the same degree.

The following sections step through each of these four search methods, explaining the relevant concepts from MCS along the way. All four of the searches eventually result in a page of search results where the user can display individual messages. Section 2.1.1 also discusses the results page in detail but the other sections omit this information since it is the same for each type of search.

2.1.1 Keyword Search

The keyword search is the primary search technique in MCS. During condensation each message in the archive is assigned one or more keywords by the editor. This type of search is very different from keyword searches in most archives where every word in a message is extracted into an index file. In these conventional archives each unique word extracted is called a 'keyword' but the keywords in MCS are potentially much more valuable. MCS keywords are chosen sparingly by the editor such that there are only a few for each message. Because the keywords are hand picked by the editor, terms that are irrelevant but appear in the message will not be promoted to keywords. The reverse is also true: the editor may decide to add a keyword to a message when that term does not appear in the message. Adding keywords which do not appear in a message is a common occurrence when a message's content implies a topic but the actual topic is never explicitly stated.

Figuring out what keyword has been used for a particular concept is a frequent problem when using conventional archives. For example, "freeze", "hang", and "lock-up" are all words that describe the same concept, but a user of a conventional archive might have to try all three in order to retrieve all the problems related to that concept. With the keyword hierarchy, the synonym problem is all but eliminated by picking one canonical term from the list of synonyms.

The keywords are organized into a hierarchy of categories by the editor. Each keyword and category can be annotated by a description and an URL when appropriate. Having a relatively

small number of keywords arranged in a tree also allows users to browse through the keywords and learn what kinds of topics are contained in the archive. Keywords can be browsed using a web interface similar to the one used at the Yahoo! web portal [14], or they can optionally be selected using a Java applet. Figure 2.2 shows the web interface for the selection of keywords.

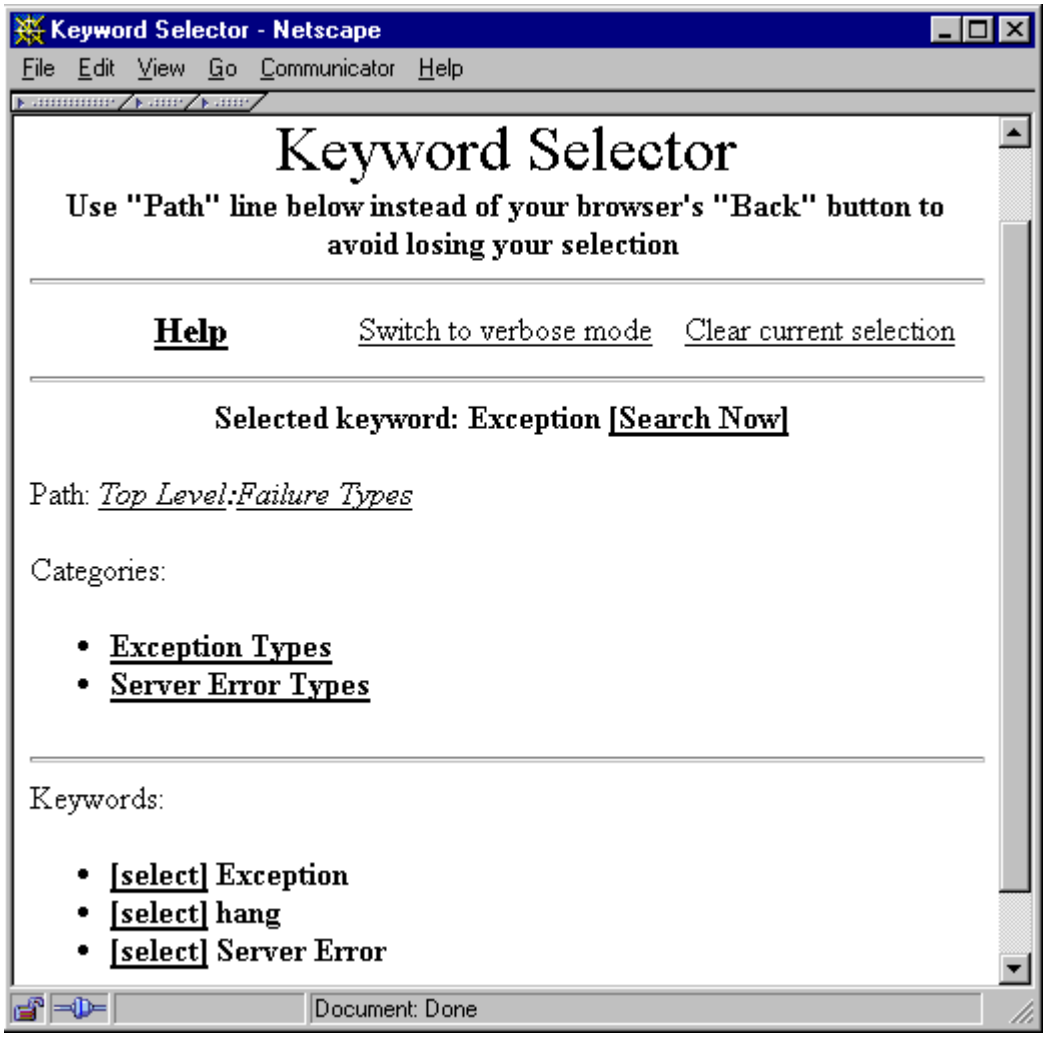

Figure 2.2. Keyword Selector interface

By choosing links in the Keyword Selector, the user can drill down into the subcategories or select one or more keywords. State information including which keywords have been selected and whether the display should be verbose is maintained in the URL. For this reason, use of the Back button in the user's browser should be avoided since that essentially 'erases' part of the state maintained in the URL. A Path bar, which shows the current location in the hierarchy, can be used

to pop back to higher levels while maintaining all state information. I chose to use URLs to save state, instead of HTTP *cookies*, because many users consider the use of cookies to be an invasion of their privacy.

After the user selects one or more keywords, the user can choose the "Search Now" link. This initiates the actual search, and displays a list of messages, which contain the keyword. If the user has selected multiple keywords, only messages which contain all of the selected keywords are displayed (a logical AND search). Figure 2.3 shows the results of a search for the keyword "Swing".

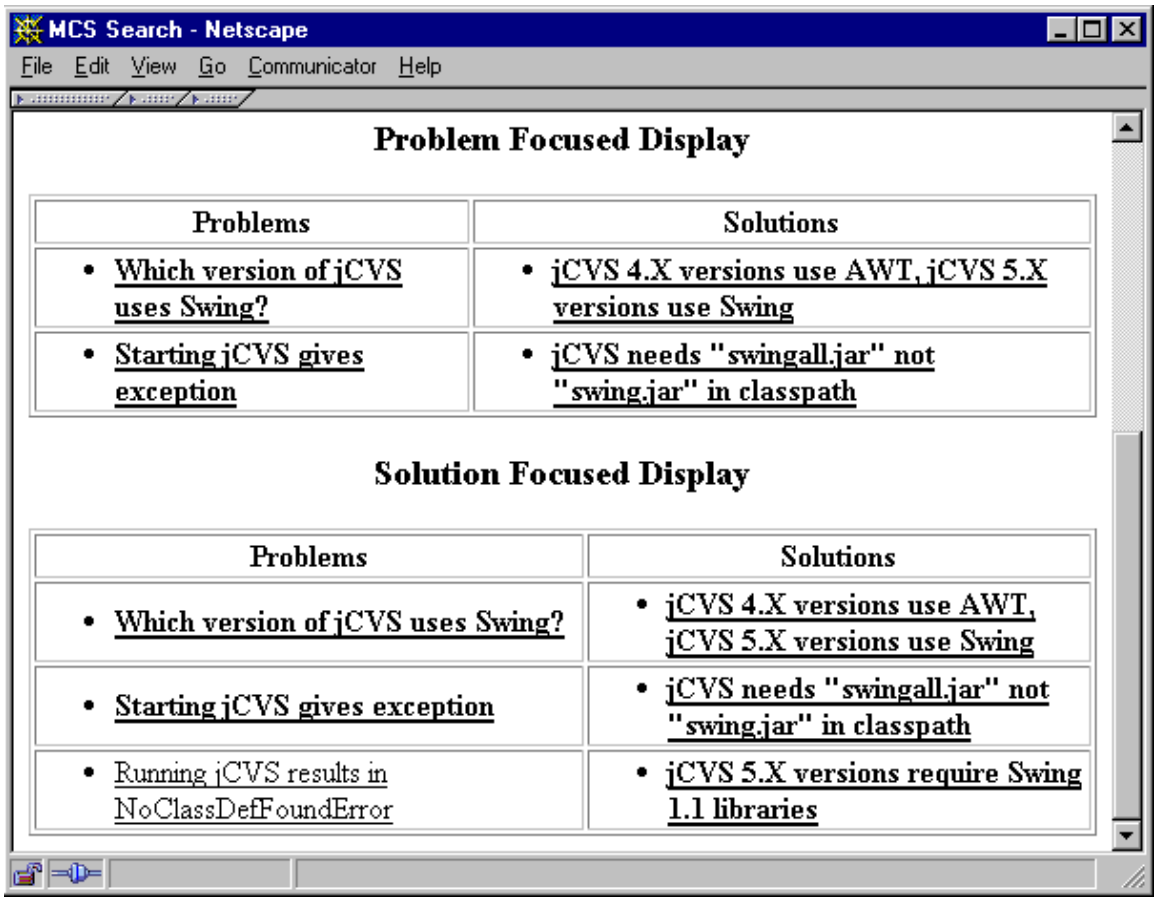

Figure 2.3. Results from a search for keyword "Swing"

Search Results

MCS's search results provide much more useful information than results from conventional archives. Instead of some cryptic message-ID or irrelevant subject line, MCS displays a brief summary of the message as a proxy for the message in search results. Seeing the summary directly

on the results page gives the user immediate feedback on whether the results are relevant to their query without having to do any further traversal.

The other major presentation difference between search results from MCS and those of conventional search engines is the categorizing and clustering of messages. Each message in the MCS database is classified as either a problem or a solution. MCS makes use of this information to segregate problems and solutions in the results display, this makes the results easier to understand. MCS also uses the links from problems to their solutions to make the relationship obvious. Each problem which matches the search criteria is displayed on the left hand side of a row in the *Problem Focused* table. On the right hand side of each row is a list of each solution which is linked to that problem. Each problem row might have zero, one, or more solutions associated with it. Following the problem-focused table is a separate *Solution Focused* table where each solution matching the search criteria has a row, with the solution on the right side and all linked problems on the left side. This somewhat complicated presentation is necessary because of the linking of problems and solutions. A problem might contain a keyword like "Unix", but the associated solution might not contain that keyword if it wasn't relevant to the solution (perhaps the solution is not platform-specific). Thus MCS's display needs to show all the matching messages plus any messages linked to matching messages. For example, a search for a particular keyword might match ten messages of which four are problems and six are solutions. However, one of the solutions is linked to a problem which doesn't contain the keyword searched for. This problem will not be displayed in the problem table, but it will be displayed across from its solution in the solution table. To indicate which messages actually matched the search criteria, messages which matched are displayed in bold while non-matching messages are not bolded. An example of this scenario can be seen in the last row of the solution table in Figure 2.3.

From the search results page the user can select any of the message summary links which will display the selected message. Figure 2.4 shows a MCS message display page. The toolbar that runs runs just above the message headers has two command buttons: "View unabridged message", and "Show all headers". The first command will display the unedited version of the message in it's entirety (Figure 2.5 shows a portion of such a display). The second command will display some hidden headers which are usually not useful for end users. The headers of the message provide meta-level information about the message:

- Filename: the name of the file where this message is stored. This is internal information displayed for debugging purposes only, and is not shown by default.

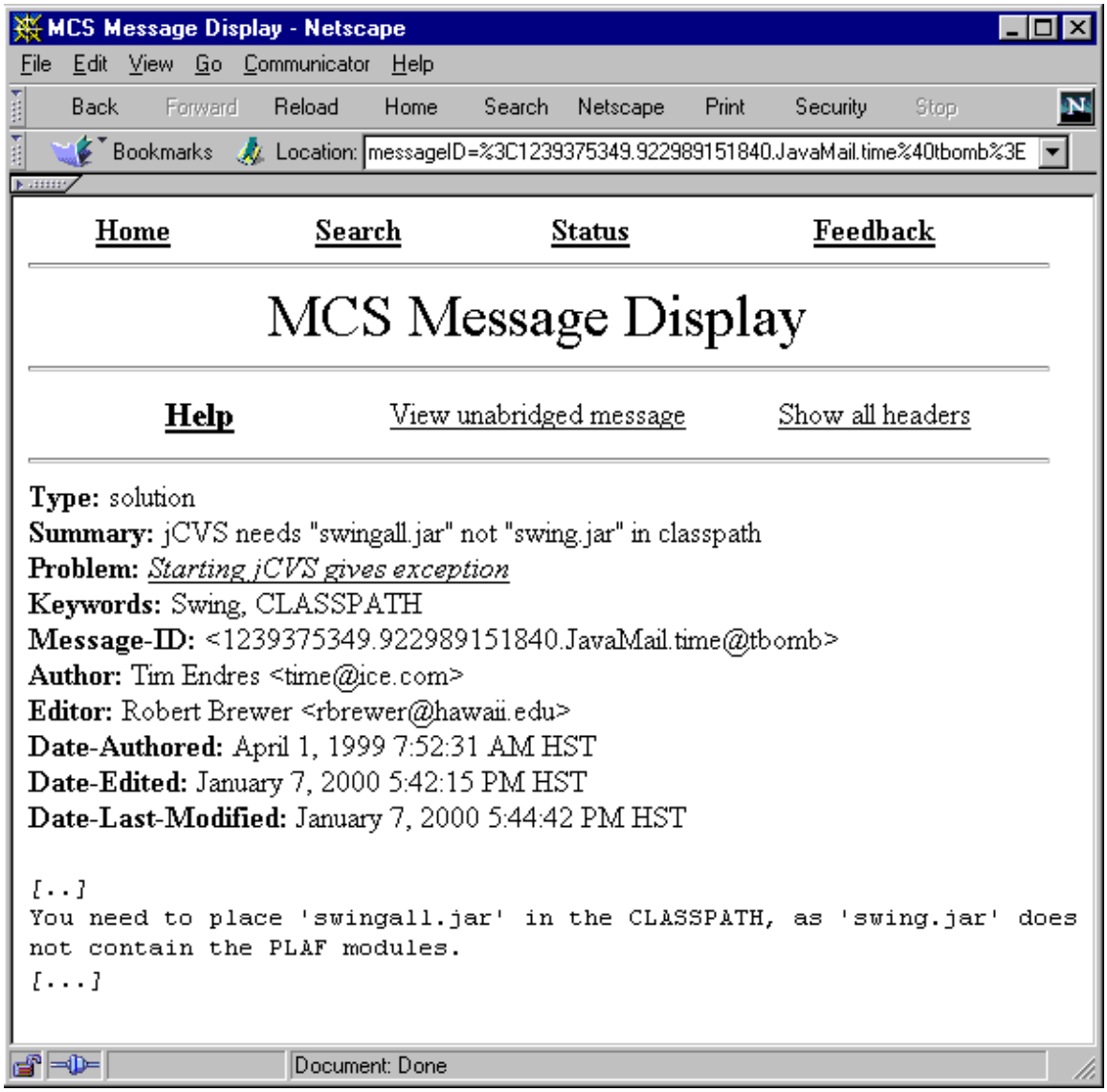

Figure 2.4. Message display page

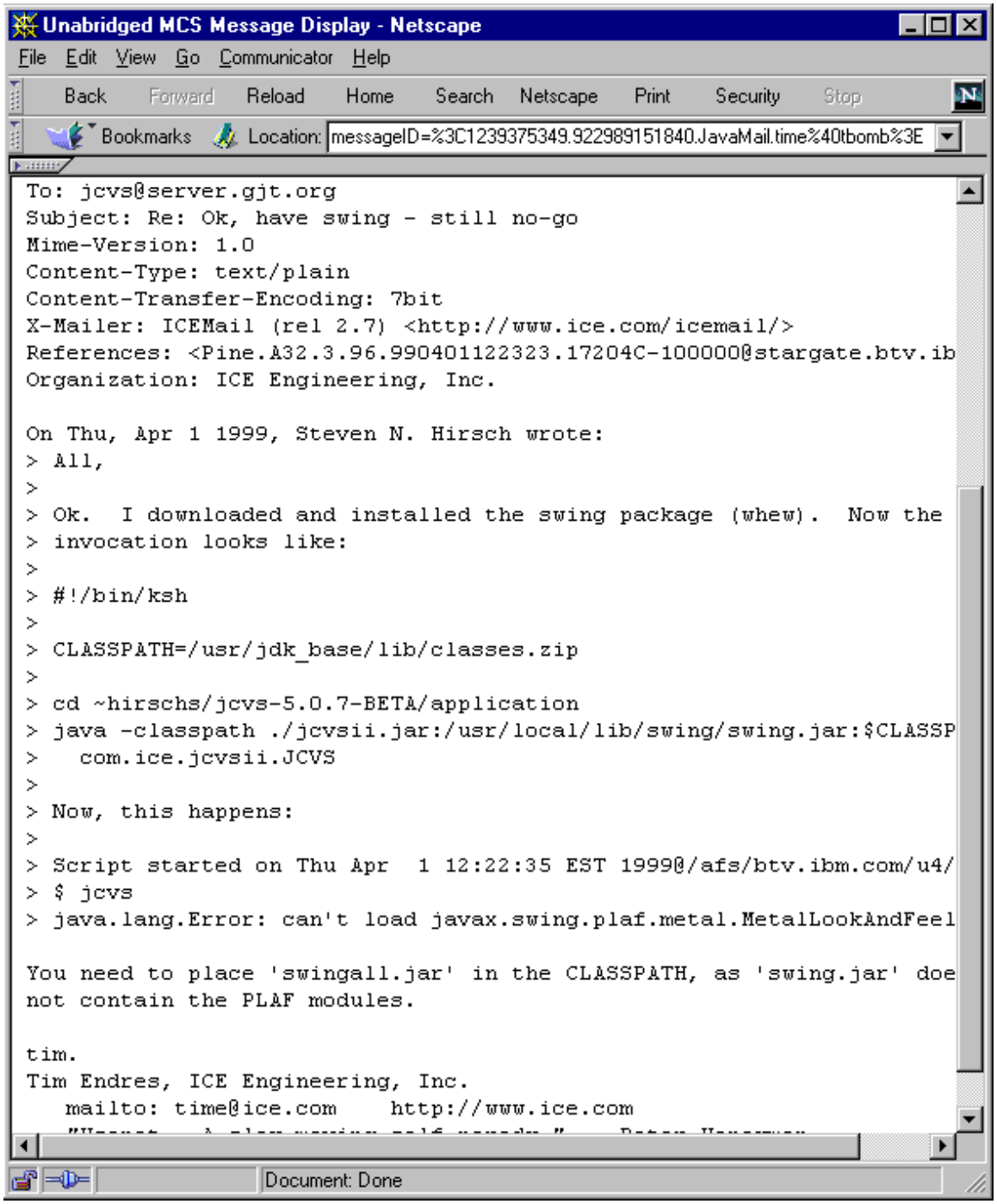

Figure 2.5. Unabridged message display page

- Type: the classification of this message, as assigned by the editor. This is either “problem” or “solution”.
- Summary: a short summary of this message, written by the editor.
- Solution or Problem: depending on the type of the current message, this header will be labeled Solution or Problem (always the opposite of the type of this message). It contains the summary of the related message(s), each of which is also a link to that message.
- Keywords: the list of keywords added to this message by the editor. It is quite possible for a message to have keywords which never actually appear in the body of the message. The keywords can include spaces and are separated by commas.
- Message-ID: the unique identifier for this message, used to distinguish messages from one another. Messages are referred to by message-ID internally throughout the system.
- Author: the name and usually email address of the original author of this message.
- Editor: the name and email address of the editor who actually edited this message.
- Date-Authored: the date and time that this message was created by the original author.
- Date-Edited: the date and time that this message was first edited.
- Date-Last-Modified: the date and time that this message was last changed by the editor. As more information becomes available to the editor, messages may be re-edited to improve their content.
- Symptom: an optional header which contains the regular expression used to match error messages to problems. Not displayed by default.
- Content-Type: an internal value which represents the MIME content type of this message. Most messages are “text/plain”. Not displayed by default.

After the headers, the body of the message is displayed. Text from the original author of the message is displayed in a normal typeface, and text that the editor has altered or added is displayed in *italics*.

2.1.2 Symptom Search

MCS also provides the capability to search for problems by symptom, as shown in Section 1.4.2. The symptom search is particularly useful because it is common for users to be aware of the symptoms of their problem, but unaware as to what the cause might be. When users encounter such a problem, they can copy and paste the error message directly into the symptom field of the symptom search web page and initiate a search. MCS will then attempt to match the given text against all the symptom expressions in the archive, displaying the results in the same manner discussed in Section 2.1.1.

The symptom search was designed to minimize the amount of effort required on the users' part. To enable users to paste in their error message verbatim, regular expressions are used to match the symptom to a problem stored in the archive. When the editor condensed the message that contained the problem, he or she extracted the symptom from the body of the message. From his or her knowledge of the domain, the editor removes parts of the error message that are specific to this particular incident (like IP address, hostname, pathname), leaving the generally diagnostic parts. The result is a regular expression that is attached to the message and stored in the archive. When a user initiates a symptom search, any line termination in the error text is removed (in case the error message was folded somewhere along the line) and then each symptom pattern in the archive is matched against the text. Any messages that match are displayed in the standard format.

The major advantage of the symptom search is that it requires minimal effort on the part of the user. However, many problems do not produce error messages so the number of problems that can be solved in this way is limited. As shown later in Section 5.1.1, only 30% of the condensed problem messages in the archive had useful diagnostic symptoms.

2.1.3 2D Search

The grouping of keywords into categories described in Section 2.1.1 enables another unique option for users. MCS allows users to perform a *2D search* by performing simultaneous searches for pairs of keywords. The user selects two categories which contain keywords using an interface similar to the one shown in Figure 2.2, and then initiates the 2D search. MCS performs the cross-product of the two categories, and for each tuple of keywords it performs an AND search of the database. The result is a table which shows the coincidence of the keywords in the two categories. Figure 2.6 shows the results of a 2D search with the categories of "Java Concepts" versus "jCVS Versions". The archive this search was performed on has a limited amount of data, but the

results provide some insight as to which concepts have proven problematic with which software versions. Note that just because the JavaHelp row has no matches doesn't mean that no messages related to JavaHelp are in the database. It just means that no JavaHelp-related messages refer to a particular version of jCVS, probably because the version of jCVS was not relevant to the problem or solution.

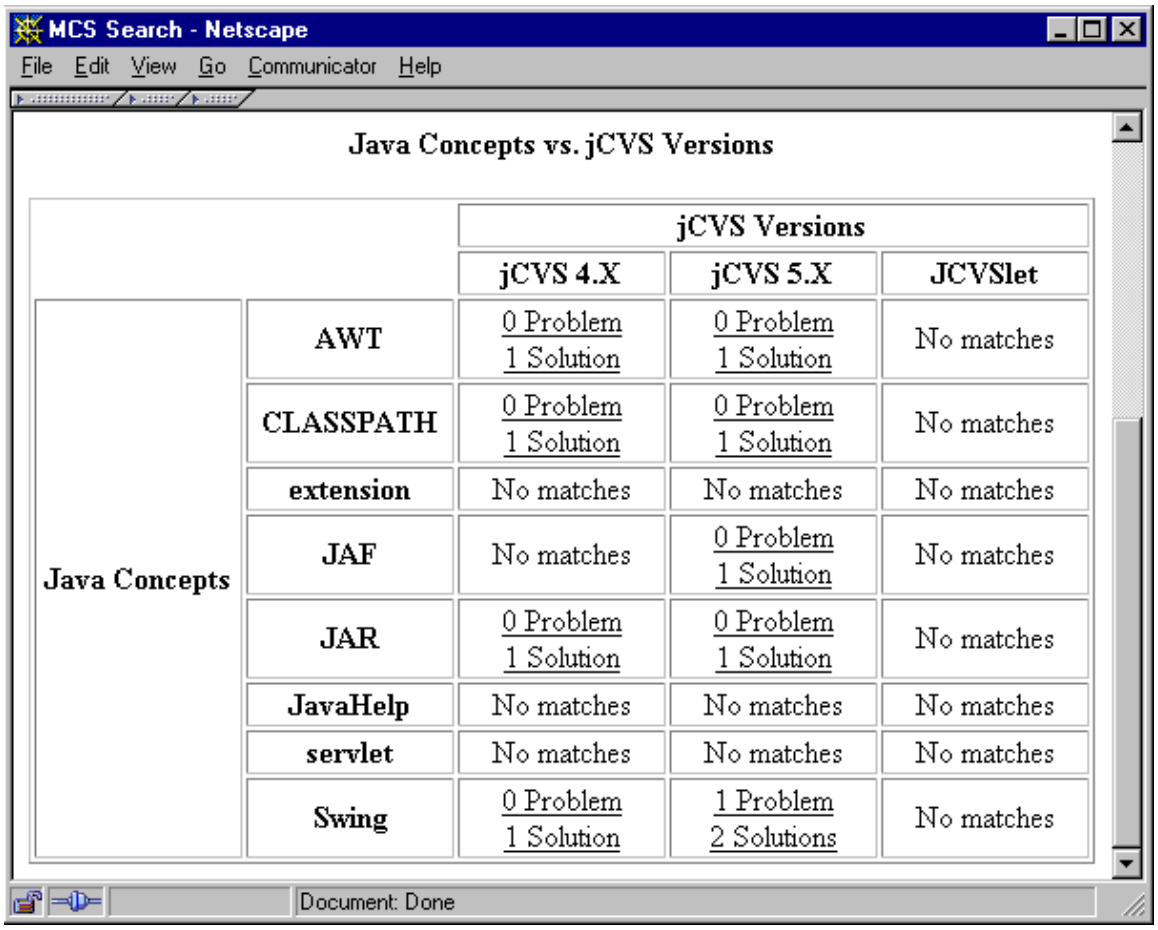

Figure 2.6. A 2D search of Java Concepts vs. jCVS Versions

Each cell of the table either shows the number of matching problems and solutions or "No matches". The cells that contain matches are also links which allow the user to see the actual matching messages from the standard MCS search results page.

The 2D search was designed to give users the ability to easily create their own views of the archive. While 2D searches are not usually the best choice when looking for the solution to a

particular problem, their ability to show trends in the data is may be useful when making decisions like which version of a program to deploy in an organization.

2.1.4 Full-Text Search

In addition to the more innovative searches previously described, MCS provides a conventional search method. All words in the body of each message are indexed (except for a standard list of stopwords like “the”) and the user can type the words they wish to search for on the full-text search page. Multiple words can be searched for simultaneously by entering them separated by commas. The words are not case sensitive but may not contain spaces. Only messages that contain all the words entered will be displayed (an “AND” type search). The full-text search is provided primarily as a last resort as in case the other types of searches fail.

2.2 The Editor Perspective

To provide the wonderful end user experience described in Section 2.1, the messages from the mailing list have to be condensed. This is the job of the editor. To make this sometimes grueling job feasible, MCS provides an editing tool which eliminates most of the bookkeeping so that the editor can focus on the higher-level tasks. The editing tool has been implemented as a set of extensions to an existing email client called ICEMail [15]. This section will step through the tasks performed by the editor.

2.2.1 Reading Messages

The first task of the editor is to become familiar with the messages to be condensed. This is done through the basic email facilities provided by ICEMail. Figure 2.7 shows this interface. The editor goes through the archive reading each message in turn, possibly deleting obviously irrelevant messages.

2.2.2 Editing Messages

The next step is to start editing individual messages. The editor selects the message in the display window and then selects the menu item “MCS edit displayed message” from the Extensions menu. This command grabs all the relevant information from the displayed message and inserts it into the message editing window which is shown in Figure 2.8.

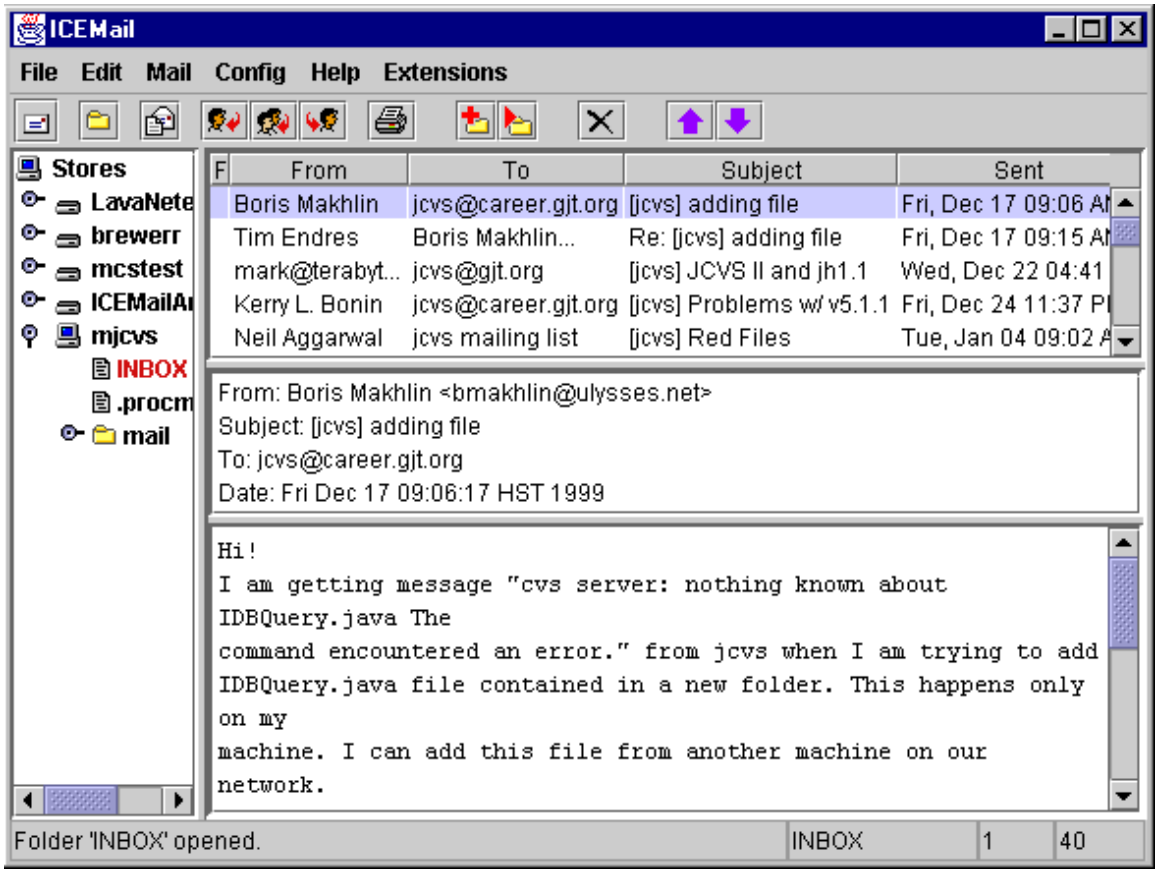

Figure 2.7. ICEMail window of editing tool

The message editing window provides a number of fields which make up the condensed message, followed by a control panel at the bottom of the window. The message fields will be described first, then the control panel.

Message Editor Fields

The “Loaded from” status field cannot be manipulated directly. It can take on one of three values:

- “Nowhere” indicates that there is no message loaded into the fields.
- “Email message” indicates that the fields are filled directly from an email message displayed in ICEMail.

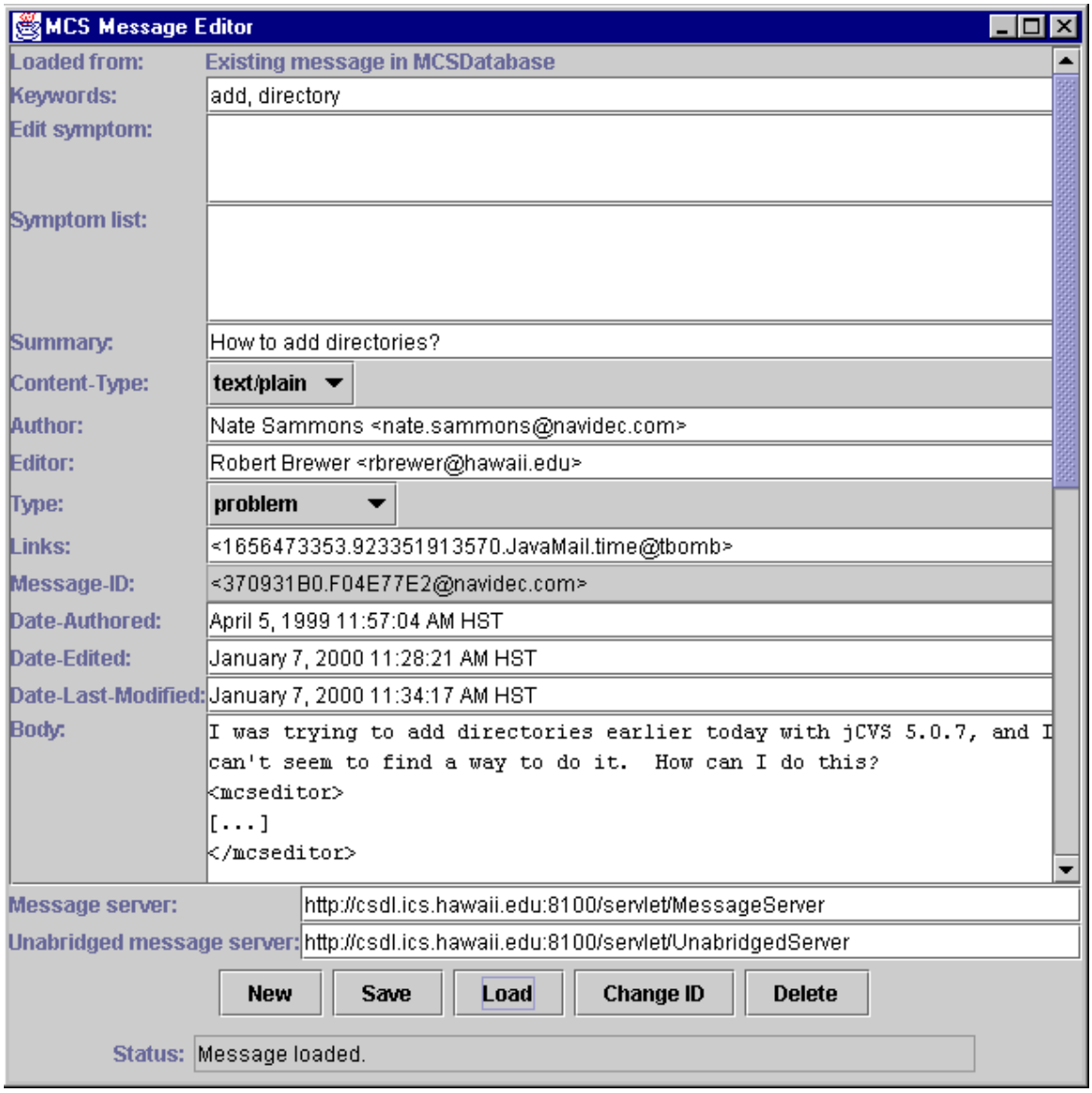

Figure 2.8. Message editing window of editing tool

- "Existing message in MCSDatabase" indicates that the message displayed had previously been saved the archive and has been reloaded for further editing.

The "Keywords" field is where the editor assigns keywords to the message. The keywords are separated by commas. Keywords can be inserted directly from the keyword editor window which is discussed in detail in Section 2.2.3. Keywords can contain spaces internally but any leading or trailing spaces are stripped.

The "Edit symptom" and "Symptom list" fields work together to allow the editor to set the symptoms attached to a message. "Edit symptom" is a text area where the editor can type in a regular expression. Suppose the editor is working on a problem message which contains same the error message used in Section 1.4.2:

```
java.lang.NoClassDefFoundError: javax/swing/DefaultBoundedRangeModel
        at com.ice.jcvsii.JCVS.instanceMain(JCVS.java:81)
        at com.ice.jcvsii.JCVS.main(JCVS.java:63)
```

The editor knows that the error message is symptomatic of using the wrong version of the Java Swing class library. So in this case, the relevant portion of this error in standard regular expression form would be:

```
java\.lang\.NoClassDefFoundError: javax/swing/.*
```

This symptom gets at the core of the error because the only two important parts are the type of the error and initial prefix of the mismatched package. The exact class which encountered the error is irrelevant as are the line numbers and file names.

Once the symptom has been entered into the field, it must be saved by clicking the right mouse button in the field which reveals a pop up menu. When the Save option on that menu is selected, the new symptom is placed into the Symptom list. If a symptom needs to be edited after being saved into the Symptom list it can be selected from that list and a right mouse click will copy it into the Edit symptom field for further editing.

The "Summary" field allows the editor to enter in a one line summary of the message. When editing a fresh email message, this field defaults to the Subject header of the email which is sometimes a good starting point for the summary.

The "Content-Type" field represents the MIME (Multipurpose Internet Mail Extensions) type of the message body. It can be set either to "text/plain" or "text/html". In the current implementation this is always set to "text/plain".

The “Author” field displays the original author of the email message. This field defaults to the contents of the From header of the loaded email message, therefore the field contains either the email address or the email address and name of the author.

The “Editor” field displays the name and email address of the editor of this message. No provision is made for recording multiple editors which might be useful if a message is re-edited later by someone other than the original author. This field is not cleared as the other fields are when a new message is loaded, to save the editor from having to continually re-enter the information.

The “Type” field assigns the message to be either a “problem” or a “solution”. It defaults to a non-valid value to force the editor to consciously choose type one or the other.

The “Links” field shows the other messages linked to this one. The other messages are represented by their message-ID and are separated by commas. Note that links are not classified themselves: anything linked to a problem is assumed to be a solution and vice versa. All links in MCS should be bi-directional (problems link to their solutions and solutions link back to their problems), but ensuring this is currently left up to the editor.

The “Message-ID” field shows the message-ID for the message. The message-ID is used as a unique identifier for each message and is used as the key in the database. Unlike the other fields, this field cannot be edited by directly selecting and changing the contents, because changing a message-ID is complicated due to its use as a key in the database. When MCS saves a re-edited message back to the archive it first deletes the existing message from the archive and then adds the new one. If the message-ID were to be changed after loading, then the original message will not be deleted and the re-edited message will be added to the archive as a duplicate. In addition, if the message-ID is changed, the unabridged version of the message must have its message-ID changed as well since the key fields are used interchangeably. This field defaults to the message-ID of the email message being edited. The Links field of messages connected to the displayed message are not changed when the message-ID is changed: they must be updated manually by the editor. To change the message-ID, the “Change ID” button must be pressed (see next section). The most common reason for changing a message-ID is to split an incoming mail message into multiple condensed messages (see Section 5.1.2 for more details).

The “Date-Authored” field show when the original email was written and it defaults to the value of the Date header of the email. This field, and the two other date fields which follow, are editable to allow the editor to correct any mistakes which might have occurred, like a inaccurate computer clock.

The “Date-Edited” field shows when the email was first edited and it defaults to the current date and time.

The “Date-Last-Modified” field shows when the email was last changed by an editor. This is useful information since messages can be changed after their initial editing. This field is automatically set to the current date and time just before a message is saved to the archive.

The “Body” field is where the actual body of the message is placed. It defaults to the body of the email message being edited. While the body is plain text, MCS recognizes certain pseudo-HTML tags in the body area. The tags `<mcseditor>` and `</mcseditor>` are used to wrap any text modified or deleted by the editor. When the message is displayed to end users, the tagged text is displayed in *italics*. The Body field also has a popup menu which allows easy insertion of the editing tags.

Control Panel

The control panel has four parts: the message server URL, the unabridged server URL, the button panel, and the status line. The two URL fields provide the location of the MCS server where messages will be uploaded and downloaded from. These URLs can be changed to point to different servers or ports depending on which archive is being worked on.

The button panel contains five command buttons which act on the message. The “New” button clears most of the fields after prompting the user for confirmation. The New button does not clear the Editor field since that field rarely changes during an editing session. The “Save” button saves the currently displayed message to the archive. If the message was condensed from an email it is saved directly. If the message was loaded for re-editing from the archive then the old message is first deleted and then the new version is saved. Along with saving the edited version of the message, the Save button sends the unabridged text of the message to the unabridged archive. The “Load” button allows the editor to reload a previously condensed message from the condensed archive, which might need to be updated. The editor is prompted for the message-ID of the message to be loaded. The “Change ID” button allows the editor to change the message-ID of a message. Changing the message-ID should be done with care for the reasons listed above in the description of the Message-ID field. The “Delete” button will delete from the archive whatever message is currently loaded and displayed. If the displayed message has never been saved to the archive then the fields are merely cleared.

The status line is used to communicate progress and status information to the editor. When commands complete they change the message in the status bar. For example, after the New button has been pressed, the status field reads “Editor fields cleared!”.

2.2.3 Keyword Maintenance

The maintenance of the keyword hierarchy is the other major part of the editor’s task. As the editor condenses each message he or she will have to decide what keywords to assign to it. The choice of what concepts are important and what actual text should be used to describe them is left up to the editor. The keywords must be selected for relevance and also for applicability to other future messages. The keywords are maintained in a hierarchy of categories which the editor also constructs and maintains. To ease this task, MCS includes a keyword editor tool. Figure 2.9 shows a keyword hierarchy displayed in the editor tool. While maintaining keywords is time consuming, having the keywords organized in this way makes browsing keywords and 2D searches available to the end user.

One of the reasons for having the editor pick the keywords rather than extracting them automatically was to avoid the synonym problem mentioned in Section 2.1.1. An alternative solution would be to maintain a database of synonyms and allow the user to search based on any of them. The problem with this solution is that it requires the editor to dream up this list of synonyms that a user might use, which is substantially harder than simply picking one term and using it as the canonical one. The MCS method does assume that a user will recognize the canonical term when he or she sees it in the keyword hierarchy.

The keyword editor window can be divided into three pieces: the keyword tree, the details display, and the control panel.

Keyword Tree

In the upper left side of the window the keyword tree is displayed. The interface should be familiar to most computer users: the top level categories are displayed and by double clicking on them their contents are revealed in an indented manner. Open categories can be collapsed as well. Only one keyword or category can be selected at any one time.

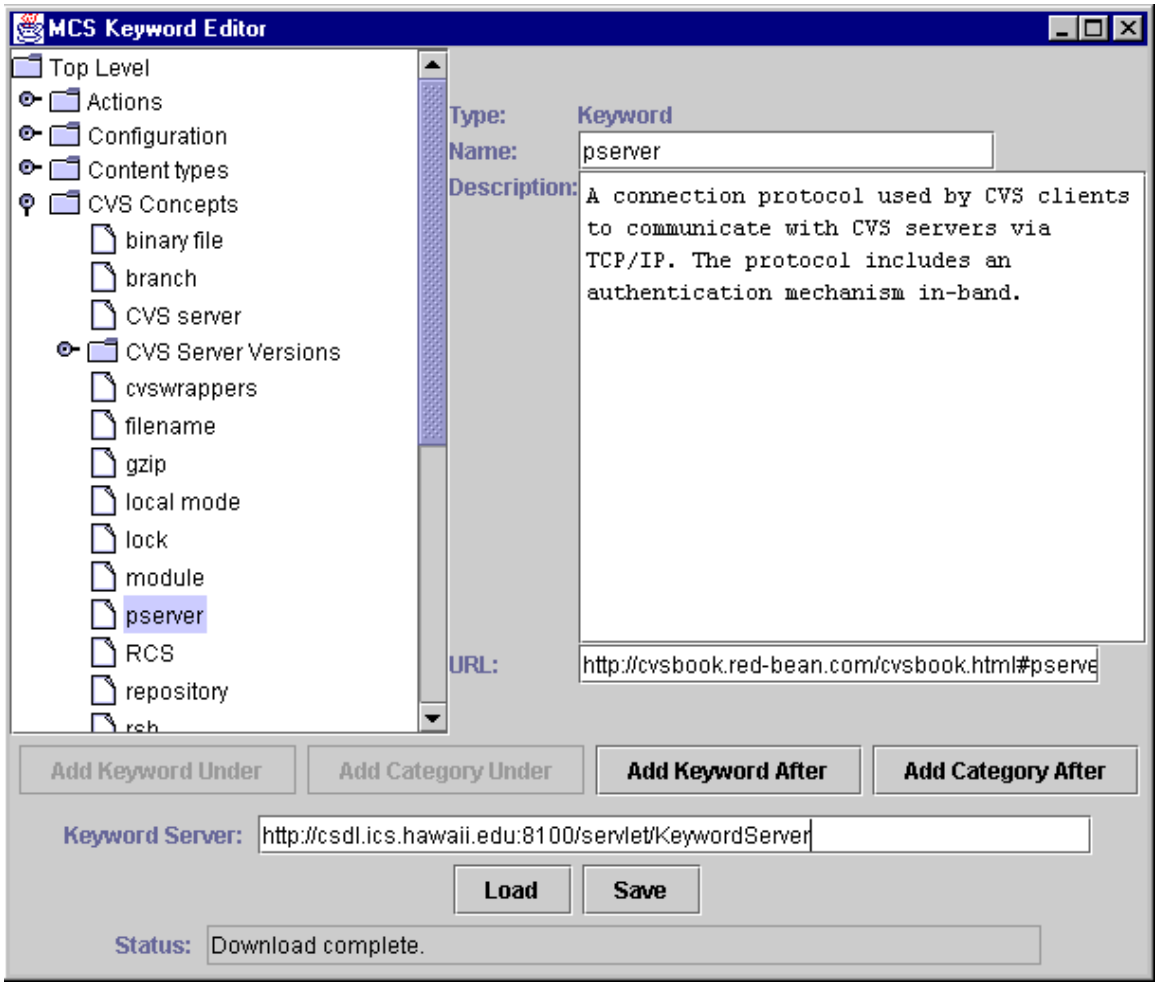

Figure 2.9. Keyword editor window of editing tool

Details Display

When a keyword or category is selected, details about it appear in the upper right portion of the window. There are four fields of information displayed for each item:

- The “Type” field indicates whether this item is a keyword or a category. This field is not editable because keywords and categories are not interchangeable. To change the type of an item, one would have to delete the item and then recreate it. I decided to make keywords and categories fundamentally different, because categories were designed purely for the classification of keywords. While it is possible that a message could actually be described by a category rather than a keyword from that category, this is unlikely in my experience as an editor. If assigning categories to messages like keywords were deemed to be a useful feature, MCS could be extended to allow categories to be assigned to messages without extensive changes.

- The “Name” field gives the name of the category or keyword. Names can contain spaces and there is no length restriction though practicality dictates that keywords not be more than 50 characters long.

- The “Description” field allows the editor to add text providing more details about the keyword or category. This can be useful if the keyword’s meaning is not immediately obvious. The description field must not contain line breaks, but the tool does not currently prevent the editor from accidentally adding line breaks.

- The “URL” field allows the editor to add a link which provides more information on the item.

The editor can make changes to any of the displayed text, but the changes are not recorded back into the keyword tree unless the “Save Changes” button is pressed. If changes are made and a different item is selected before saving the changes then the changes will be lost.

Control Panel

The control panel contains all the command buttons. The first row contains seven buttons:

- The “Add Keyword Under” button creates a new keyword inside the currently selected category. The new keyword is created blank with the name “New Keyword”. This button is disabled when the item selected is a keyword since keywords cannot contain other items.

- The “Add Category Under” button creates a new subcategory inside the currently selected category. The new category is created blank with the name “New Category”. This button is disabled when the item selected is a keyword since keywords cannot contain other items.

- The “Add Keyword After” button creates a new keyword just after the currently selected item at the same level in the hierarchy. The new keyword is created blank with the name “New Keyword”. This button works when either a keyword or category is selected.

- The “Add Category After” button creates a new category just after the currently selected item at the same level in the hierarchy. The new category is created blank with the name “New Category”. This button works when either a keyword or category is selected.

- The “Save Changes” button allows the editor to commit any changes he or she has made to the text in the three editable display fields. If another item is selected before pressing this button then any changes to previous item will be lost.

- The “Insert” button allows the editor to insert the currently selected keyword into the message editor window. This makes it easy for the editor to assign keywords to a message by just browsing the keyword tree and inserting relevant keywords. The button ensures that the keywords inserted into the message editor will be properly comma separated. This button is disabled when a category is selected because categories cannot be assigned to messages.

- The “Delete” button deletes the currently selected item from the tree. There is no user confirmation.

The second row of the control panel contains the controls for loading and saving the keyword tree back to the archive server. Keyword tree I/O is done separately from the message editing and the keyword tree could conceivably even be stored on a separate server. The “Keyword Server” field contains the URL of the KeywordServer servlet which connects the editing tool to the archive. When the “Load” button is pressed, the keyword tree is downloaded from the server and displayed for editing. The user is reminded via a confirmation dialog box that this will erase any unsaved changes in the currently displayed keyword tree before continuing. The “Save” button writes the keyword tree back to the server.

The third row of the control panel contains the status line. This line is used to display progress messages to the user during various operations and to indicate when operations fail.

2.2.4 Archive Testing

As the messages are being condensed and saved, the editor will want to make sure that messages are appearing properly in the archive, and that the links between messages function as expected. There is no special interface provided to the editor for testing: the editor simply accesses the archive as an end user would.

Chapter 3

MCS System Architecture and Design

MCS is the system I have created that implements the idea of condensation. This chapter describes the underlying requirements, architecture, and implementation details of the system.

3.1 Requirements

MCS was designed to help users of problem-solving mailing lists by improving the usability of their archives. Making archives more useful not only helps the archive users, it also helps to improve the quality of the mailing list itself, because people are less likely to re-request information which is easily available via the archive. To achieve this goal, the user community must adopt MCS in preference to the many existing systems for generating and maintaining searchable mailing list archives. To encourage users to adopt the system, the design of MCS takes into account two issues: an explicit domain focus, and the existing list community.

Most mailing list archive search engines are designed to work with any mailing list. Because they must work with any mailing list, conventional search engines are limited to keyword searches and simple search results presentation. The idea of MCS is the exact opposite: mailing list archives can be enhanced by tailoring the search engine to a particular mailing list domain. By embracing the details of a particular kind of mailing list MCS provides greater utility and efficiency for archive users.

To explore the idea of domain-specific enhancements, I had to choose a domain. The domain I selected was product support mailing lists dedicated primarily to problem solving. The focus on a single domain simplified both the implementation and the execution of the case study. Because these lists focus on problem solving, the users of their archives are also interested in problem solving. For this reason, the MCS-condensed archive contains only messages describing problems or

solutions. Users looking for a comprehensive archive of all messages sent to the list, can consult an existing traditional archive. The existence of other "unabridged" archives frees MCS to eliminate any messages or parts of messages that are not worth archiving.

Because MCS receives its input from a mailing list, it is crucial that MCS be designed with the social structure of a user-supported mailing list in mind. Specifically, the mailing list and its community should not be adversely affected by MCS. Any attempt to impose restrictions on how people read or participate in the list (like requiring users to use special software or compose messages in a certain format) would be met with blistering criticism. MCS must stand apart from the mailing list itself, limited to using messages from the list on an as-is basis. MCS also takes into account the needs of the user community by having very low requirements accessing the archive. The archive is accessed using a web browser which is presumably standard equipment for most mailing list participants. Furthermore, the web pages themselves are simple; they contain no images, no Java applets, and no JavaScript to ensure that users can use older browsers to access the archive. The omission and editing of messages is central to MCS, but those actions can reasonably arouse suspicion among list members as to the fairness of the editing. To assuage these fears and to assure context, MCS provides a link from each edited message to the original message maintained in a separate unabridged archive.

The editing process itself imposes certain constraints on the system. Since a human does the editing, the amount of effort required to condense a message must be kept fairly small. The system would not be useful if the overhead of condensing anything other than a trivially small archive is too great. One solution to the problem of editing overhead is to spread the work out over multiple editors. However, I designed the current system on the assumption that only one editor will be condensing at any time. Adding support for multiple editors is possible and discussed further in Section 7.2.1.

To support future goals of open source distribution and adoption by many mailing lists (see Section 7.2), MCS must be portable. Mailing lists and their archives operate on a broad spectrum of platforms so MCS must impose as few constraints as possible in that area. It should also be self-contained to encourage people to install it.

3.2 Architecture

In this section I discuss the overall architecture of the MCS system. The users of MCS can be broken down into two different roles:

- Editor. The editor actually performs the condensation of the archive. In the current system the editor is a human, but in the future this role could conceivably be taken over by a sufficiently advanced AI system.

- Archive user. Archive users are the people who actually make use of the condensed archive.

From these two roles springs the basic division of the system. It consists of three subsystems:

- A centralized database which stores the messages that form the archive, and supports queries on that data.

- An editing tool to condense the queue of messages from the mailing list which are then fed into the central database.

- Display and query tools which provide an interface to the database for archive users.

The architecture of MCS is comprised of several blocks which are shown diagrammatically in Figure 3.1.

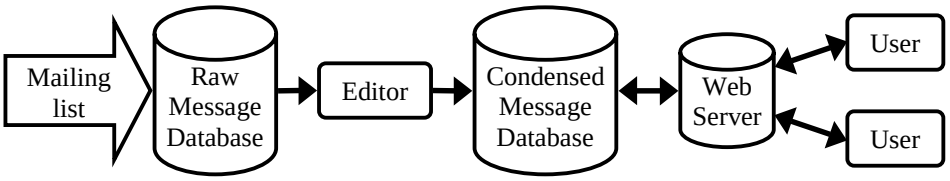

Figure 3.1. Block diagram of MCS system architecture

The input for MCS obviously comes from the mailing list itself. The central server is subscribed to the mailing list and receives each message as any subscriber would. The editor then uses the editing tool to pick up the messages waiting in his or her queue and condenses them. The condensed messages are shipped back to the central server which stores them in a database. The editing tool also allows the editor to create and maintain the keyword hierarchy.

Users access the archive using a normal web browser. Users can select from one of the four searching methods, and then enter their search parameters. The system then performs the search and returns a list of matching messages.

3.3 Design

I implemented all executable aspects of MCS entirely in Java 2 (also known as Java 1.2). This decision follows the requirement that MCS (server and editor) be usable on a wide variety of platforms. I implemented the end user interface in HTML as web pages, thereby enabling almost universal access by archive users.

MCS consists of three different subsystems coded in Java: a cluster of servlets (a way to extend the functionality of servers, particularly web servers) which provide the storage and retrieval of the messages, two extensions to the ICEMail program which make up the editing tool, and two Java applets which are used as an experimental searching mechanism.

The code is broken into five different packages: `csdl.mcs.data`, `csdl.mcs.editor`, `csdl.mcs.gui`, `csdl.mcs.util`, and `csdl.mcs.web`. Figure 3.2 shows the packages and their calling relationships.

3.3.1 Package `csdl.mcs.data`

This package is the glue that holds together all the other packages. It contains all the abstract data types used in the system like `MCSMessage` which represents the individual messages in the system, and `KeywordTreeNode` which represents each item in the keyword tree. The other major part of this package is the `MCSDatabase` object. `MCSDatabase` is a static class which provides an abstraction layer for the actual data in the archive. All the servlets search and retrieve data from the archive through calls to `MCSDatabase` to allow for future changes to the underlying data storage and indexing implementation. The class needs to be static because all the servlets require access to the information but they don't have access to each other's data. The static class provides a shared data space that all servlets can access because they share the same execution environment.

3.3.2 Package `csdl.mcs.gui`

This package contains only one class: `KeywordTreePanel`. This class displays the keyword tree and the details of the selected item. This class is used by the keyword editor and the Java searching applets because they both need to display the keyword tree.

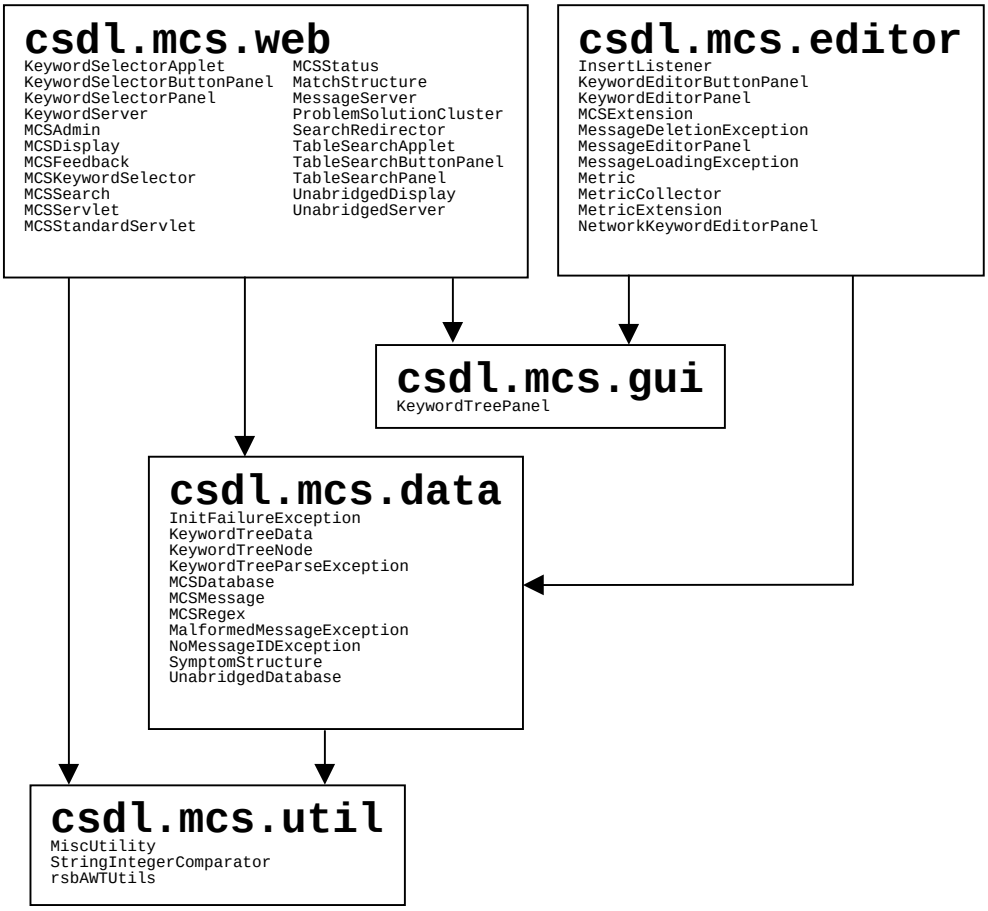

Figure 3.2. Diagram of package relationships in MCS. Some regression testing classes have been omitted.

3.3.3 Package `csdl.mcs.editor`

This package contains the editing functionality of MCS. The two major classes are `MessageEditorPanel` and `NetworkKeywordEditorPanel` which are used to create the two editing windows (see Figures 2.8 and 2.9). The panels are built out of a layering of components. For example, `NetworkKeywordEditorPanel` adds functionality on top of `KeywordEditorPanel` which is an amalgamation of `KeywordTreePanel` and `KeywordEditorButtonPanel`. The package also contains some modest wrapper classes which turn the panels into ICEMail extensions.

3.3.4 Package `csdl.mcs.util`

This package contains miscellaneous utilities used by the other packages. Most of the methods are static as they are intended to be used directly without the need to instantiate the class. For example, the `MiscUtility` class provides methods for escaping HTML metacharacters, finding the intersection of two Vectors, and counting the number of elements in a comma delimited String.

3.3.5 Package `csdl.mcs.web`

This package contains all of the servlets used to run the archive and the search applets. Some of the servlets like `MCSSearch` and `MCSDisplay` provide the interface to the archive user.Another group of servlets, `KeywordServer` and `MessageServer` provide communication services to the editing tool. All the servlets conform to version 2.0 of the Sun Servlet API, which means that they are compatible with any of the many web servers which support servlets.

3.4 Implementation

Since MCS is a research project, the implementation priorities were different from a commercial package. Two of the major priorities were: creating an accurate implementation of the condensation concept, and completing the implementation as quickly as possible. While I gave some thought to future expansion, I spent very little time optimizing MCS's execution speed or memory requirements. There is little point in spending time optimizing a system before it has even been demonstrated that the system is useful. However, the performance of MCS has been more than

satisfactory in my usage to date. This section provides a brief tour of the MCS implementation by following the life-cycle of an archive.

3.4.1 Startup

The core of MCS is the group of servlets and support classes that run within a web server. All the archives created so far use Sun's Java Web Server, but any web server that supports servlets and embedding servlet output into static HTML pages could have been used. All the servlets and their support classes are bundled together into a single Java archive file (JAR file) which is added to the classpath of the web server. I discovered several incompatibilities in the Java execution environments available to me which prevented the proper operation of the servlets. Finally I found that the Solaris Reference JDK 1.2.2 with the HotSpot VM 1.0.1 worked both with and without the JIT compiler enabled (debugging is easier when the JIT compiler is off since line numbers are provided in stack traces). The MCS archive also needs an HTML start page that contains special server side tags to embed the output of the `MCSStatus` servlet in the page.

As mentioned in Section 3.3.1, all the servlets communicate via the `MCSDatabase` object which has only static methods and member variables. However, the `MCSDatabase` needs to read in information about the database from disk before its methods are called. I solved this problem by putting special initialization code in one servlet's `init()` method which reads initialization parameters passed by the web server. These parameters are then passed on to `MCSDatabase` so that it can properly initialize. To ensure that `MCSDatabase` is initialized before the web server accepts any HTTP requests, the web server was configured to load the special servlet when starting up. I chose to make the `MCSSearch` servlet the special servlet, since it was likely to be used as soon as the server starts. The initialization parameters (like the directory where messages are stored) are configured in the web server, so no archive-specific information has to be compiled into the servlets.

`MCSDatabase` initializes by building several internal data structures:

- A Hashtable mapping from message-IDs to filenames
- A Hashtable mapping from a keyword to a Vector of message-IDs
- A Vector of symptoms and their associated message-IDs
- A Hashtable mapping from full text words to a Vector of message-IDs
- A Hashtable containing all the stopwords (words not indexed in the full text Hashtable)

All these data structures are built by scanning every message in the archive. This process is fairly slow, but only happens at startup (or as a result of an infrequent action). This process could easily be optimized by building the indexes and saving them as separate files instead of scanning each message. MCS keeps all this meta-level information in memory which might not scale well for very large condensed archives, but does make for fast queries on small archives. In addition to this meta-level information, `MCSDatabase` reads in the keyword hierarchy and the last modification date for use by the servlets.

A parallel class, `UnabridgedDatabase`, for unabridged messages requires similar initialization. However the class is much simpler, because it only supports retrieval by message-ID, but no searching methods.

3.4.2 Editing

The editing tool makes use of the fact that the data source for condensation comes in the form of email. The messages to be condensed are stored in normal mailbox folders on an email server. The Java email client ICEMail [15] has been extended for use as the MCS editing tool. Since ICEMail is an open source program, I added hooks to it that cause it to load extension classes. The names of the extension classes are added to a property in the ICEMail configuration file. ICEMail extensions implement a simple interface which provides for initialization and shutdown of the extension. On startup, ICEMail reads the names of the extension classes, loads them via the standard class loader, and then calls their `init()` method.

There are two ICEMail extensions in MCS: `MCSExtension`, and `MetricExtension`. `MCSExtension` provides access to both the message and keyword editors. `MetricExtension` collects and records low-level data about the editor's activities for analysis.

The keyword editor loads and saves the keyword tree via HTTP connections to a servlet named `KeywordServer`. To receive the keyword tree, the keyword editor connects to a given URL and sends an HTTP GET request and the servlet responds with the keyword tree flattened into a textual format (the same one used to store the keyword tree to disk) which it has retrieved through a call to the `MCSDatabase` object. The keyword editor then parses this into the internal format and displays it to the user. When the editor wants to save the tree, the keyword editor window sends an HTTP POST request and sends the keyword tree in the textual format. The servlet then passes this information on to the `MCSDatabase` object which stores it to disk. There is no provision for partial updates of the keyword tree, and there is no locking protocol. In addition, no authentication or authorization is performed on these operations so the only security is provided through obscurity.

This could easily be remedied through the web server's HTTP authentication mechanisms which are straightforward to implement on both the servlet and editor side.

The message editor is a little more complicated than the keyword editor. The message editor tool must deal with both the edited and unabridged versions of each message. There are three operations that the message editor can perform on the database: save a message, load a message, and delete a message. When a message is saved in the message editor window, it makes HTTP connections to both the `MessageServer` and the `UnabridgedServer` using the URLs provided by the user. The two connections are made in parallel, because if the upload to one servlet fails, the other upload should be aborted to avoid an inconsistent archive state. The upload is made as an HTTP POST request in a textual format, which the servlet reconstitutes and sends to either `MCSDatabase` or `UnabridgedDatabase` as appropriate. The static object writes the new message to disk and adds the meta-level information to the in-memory data structures.

To load or delete a message the message editor makes an HTTP GET request to both servlets. It passes two parameters to each servlet: message-ID and function. The message-ID indicates which message is to be acted upon and the function indicates whether to load or delete the message. When loading, the servlets will request the message from `MCSDatabase` or `UnabridgedDatabase`, convert it to text, and send it to the message editor. Deletion of an edited message is complicated by the in-memory meta-level information which needs to be updated. The current implementation solves this problem by simply deleting the message file from the disk and then forcing `MCSDatabase` to rebuild all the meta-level information from scratch! While this kludge simplifies the implementation mightily, it makes message deletion a time consuming operation. On an archive of 177 messages the rebuild can take 5 to 10 seconds, but message deletion happens only infrequently, making it acceptable. To optimize this operation all the meta-information tables could be stepped through and all references to the message-ID removed.

`MCSDatabase` and `UnabridgedDatabase` are write-through for all editing operations, so there is little chance for data loss in the case of a server crash.

3.4.3 Searching

End users of the MCS-condensed archive access the archive using their web browser. The front page of the archive is an HTML page with an embedded call to the `MCSStatus` servlet, so that users can immediately see how many messages are in the archive and when the archive was last modified. There are also a variety of help pages which are written in static HTML. However, the rest of the web pages seen by the user are created dynamically by the servlets in the system. Each

servlet can be accessed directly via an URL like <http://csdl.ics.hawaii.edu:8100/servlet/MCSStatus> and they will generate a full page of HTML which is displayed by the browser. The MCSSearch and MCSStatus servlets optionally take an "embedded" parameter which causes them to omit header tags, which are not needed when the output is spliced into an HTML page.

The servlets communicate with each other in two ways: the static MCSDatabase object, and parameters in URLs. The MCSDatabase object contains the information which relates to the archive itself, so most inter-servlet communication is done via URLs. As mentioned in Section 2.1.1, the URL is also used to save state between servlet invocations. I needed to use this technique because HTTP is a stateless protocol.

Most of the operations the searching servlets perform eventually boil down to calls to MCSDatabase or UnabridgedDatabase. For example, the MCSDisplay servlet takes two URL parameters: messageID and verbose. Most of the code in this servlet involves writing HTML headers and doing error checking on the parameters. The real work of retrieving a message from the database is done by a call to MCSDatabase.getMessageFromID() which returns a MCSMessage object when given a message-ID. The MCSSearch servlet, however, contains a substantial amount of code to perform the four different types of searches. To allow initiation of searches and display their results from the same page, the MCSSearch servlet is 'reentrant'. When the user initiates the search from the form generated by MCSSearch, the form data is sent via the URL to the MCSSearch servlet. This is another example of the servlet state being maintained in the URL. Most of the code in MCSSearch involves the clustering and display of the search results it receives from method calls to MCSDatabase.

3.4.4 Implementation Metrics

The implementation of MCS consists of 7387 lines of Java code, 343 methods, and 55 classes. On the server side, MCS uses 13 HTML files (primarily help files) which total 41 kilobytes in size. I implemented the system in several stages as the various research ideas were explored. First I implemented the archive servlets to test the basic research idea. The archive used a small number of messages which I had condensed by hand. Next I implemented the keyword selector and editor to allow keyword searches. Then I implemented the 2D search method on top of the keyword selector. Finally I wrote the code for the message editor and ICEMail integration, which allowed me to condense an archive of useful size. It took me 18 months to complete the implementation.

Chapter 4

Case Study Design

To evaluate the research hypothesis that MCS is an improvement over existing archives, I designed a case study of MCS. The case study involved creating a condensed archive of a mailing list, releasing the archive to the list subscribers, and collecting qualitative and quantitative data on the users of the archive. This chapter details the design of the case study and the type of data that I collected. Section 4.1 discusses the characteristics of the mailing list I used in the case study. Section 4.2 explains the three different measures which I used to evaluate the research hypotheses. Section 4.3 details the three different types of data collected in the study and how I analyzed them. Section 4.4 lists the stages of the study implementation.

4.1 Target Mailing List

To perform a case study, I had to select a mailing list. As mentioned in Section 1.4.2, I selected the jcvs mailing list. Appendix A shows the subject lines from a few days of traffic on the list. The jcvs list has the following characteristics:

1. The main topic of the mailing list is solving problems encountered in jCVS. While there are some other discussions like feature requests and interface design reviews, the bulk of the messages are either descriptions of problems or solutions to those problems. There is little or no traffic of a purely social nature.

2. Most of the list subscribers are developers who are notoriously busy people. This fact works to our advantage for two reasons. First, because they are short on free time, their willingness to adopt a new system will in itself be an accomplishment. Second, they are likely to judge the system primarily by whether it helps them get their job done faster or not.

3. I am familiar with the list subject matter of software development and version control. As the editor of the jcvs condensed archive, I required knowledge of the mailing list topic in order to properly categorize information.

4. The list had 401 subscribers at the start of the case study, which is large enough to provide a pool of archive users.

5. The mailing list already has an archive which allows users of the condensed archive to compare it to the existing archive.

6. The list is relatively low traffic. There are roughly 15-20 messages sent to the list a week, which provided sufficient input to the condensation system without overwhelming the system or its maintainer.

7. 1428 messages had been sent to the mailing list by the time the condensation had completed. This allowed me to create a condensed archive of non-trivial size without requiring an excessive amount of time for condensation.

8. The author of jCVS and the jcvs list maintainer (Tim Endres) stated, on the mailing list on January 19, 1999, that the jCVS package was downloaded over 20000 times per year. At that time, he estimated that jCVS was used "pretty actively" by over 1000 sites.

9. Tim was very enthusiastic about the MCS project and was eager to provide the number of subscribers and the existing list archives in a format usable by the editing tool. This support made the condensation feasible.

Some of these characteristics are also requirements for any list to be condensed by MCS, and some are specific to the practicalities of this research project. Characteristic 1 is necessary for any MCS list archive because MCS is geared towards problem solving lists. Characteristic 3 is also important since the editor needs to know the subject matter to properly condense. Characteristic 9 is a practical one: you cannot condense a list if you don't have access to the archive and permission to use it. The rest of the characteristics relate primarily to the evaluation of the system, and are, therefore, not requirements for future MCS targets. In fact, larger mailing lists would presumably benefit even more than smaller ones if the editing overhead can be met.

It is also worth noting that the jcvs mailing list is hardly the only list which meets the essential requirements. Several other lists like the previously mentioned bsdi-users, or the ascend-users lists are good candidates. The main problem with those two lists is that they have been in

existence for several years so their archives are voluminous. Condensing those archives would be infeasible for this research project due to the limited resources available (one graduate student). However, these lists [16] could be good candidates for future condensation (see Section 7.2).

4.2 Evaluation Factors

In this section I discuss the three measures used to test the thesis statement presented in Section 1.5: editor overhead, adoption percentage, and MCS preference percentage.

4.2.1 Editor Overhead

Even if condensed archives are deemed by users to be more useful than conventional archives, the system will not be adopted if the condensation process requires too much effort by the editor. The editing task must be feasible to perform, and it must not require too much time spent per message. I assessed this measure by collecting time data from the editing tool as the archive was condensed, and by introspecting about the editing process.

4.2.2 Adoption

To measure the adoption percentage defined in 1.5, I needed two pieces of information: the number of list subscribers and the number of users of the condensed archive. The list maintainer provided the number of list subscribers from the subscription list. An estimate for the number of users of the condensed archive was obtained by surveys and analysis of log data from MCS. Note that the adoption percentage, as I have defined it, is an imperfect measure since I cannot positively determine whether the users of the condensed jcvs archive are actually subscribers of the mailing list.

4.2.3 Preference

Assessment of the users' preference of the condensed archive over traditional archives was determined in a qualitative way through user surveys. My survey included the question: "Since the new problem-solving archive has been available, do you find yourself using it instead of the old archive?".

4.3 Data Sources

The case study was designed to provide data from four different sources: editing experiences, in-person demos, web server logs, and a questionnaire. In the following sections I describe: each source of data, how the data was collected, and how it can be used to compute the measures in Section 4.2.

4.3.1 Editing Results

To gain insight into whether condensation is feasible for editors, data was collected about the editing process. I recorded data on both the time required to condense a series of messages, and how many condensed messages and keywords resulted from the process. Since there is only one editor, qualitative feedback on editing is also valuable source of information. I kept a diary of issues encountered while condensing the archive which is detailed in Section 5.1.2.

4.3.2 In-Person Demos for Potential Archive Users

To evaluate the design and usability of the system before it was released to the list subscribers, I gave several demos to colleagues in the ICS department. I gave the first set of demos to the Collaborative Software Development Laboratory (CSDL) of which I am a member. After I had implemented some of the suggested changes, I demoed the archive to a member of my committee to get his assessment as someone who was not close to the development of the system.

4.3.3 Web Server Log Analysis

The Java Web Server, like most web servers, records a log of all HTTP [17] requests made to it. Each log entry contains the request made, the IP address of the requester, and a timestamp. At this level, the data provides mere "hit count" information which is a poor indicator of the number of actual users of the system. To track the number of users, I counted the number of unique IP addresses making requests. This technique, however, has problems because of dynamic IP addressing and the use of public access computers (such as in a University lab) [18]. In the case of dynamic IP addressing, a user may access the archive from the same computer but over the course of a day that computer's IP address might change which would cause this user to be counted more than once. In the case of a public access computer, multiple people may use the same computer to access the archive. Since the computer only has one IP address, the multiple users will only be counted once.

Despite these inaccuracies, counting by unique IP address should provide a rough estimate of the number of users. Other more accurate alternatives for counting visitors exist; however, they require the user to either register and log on or accept *cookies*, which many people consider intrusive. Since the major goal for MCS is adoption, annoying users is to be avoided at all costs. Dynamic IP addressing is expected to be more prevalent than shared computers among jcvs subscribers, so I expect the unique IP address count to be an overestimate of the adoption rate.

Another method for estimating the number of users of the archive is organizational analysis. Organizational analysis attempts to collate the number of distinct organizations that issued requests to the web server. While requests are recorded in the log by IP address, the Domain Name System (DNS) can be used to map the IP address into a domain name. Domain names can more more useful than raw IP addresses as they indicate what organization an IP address belongs to. Organizational analysis categorizes the requests according to domain name. However, determining which portion of the domain name indicates a unique organization requires additional processing. Top Level Domains (TLDs), such as ".com" and ".uk", use different organization schemes. For example, most organizations in the US are uniquely identified by the two top levels of a domain name: "collab.net", "lanl.gov". Some other countries require three levels of hierarchy before reaching the organizational level: "monash.edu.au", and "demon.co.uk". Using a lookup table based on the TLD, the portion of the domain name which indicates a unique organization can be determined. Then all requests originating from that organization can be categorized together. This analysis results in a list of organizations which accessed the archive, which is much less sensitive to the problems of counting raw IP addresses mentioned in the previous paragraph. Of course this method has its own problems: multiple users at the same organization are only counted once, and some IP addresses cannot be resolved to a domain name. However these problems should make the size organization list an underestimate of the number of users of the archive.

4.3.4 Brief User Questionnaire

To obtain broader feedback on the system, I administered a web-based questionnaire. I considered distributing the questionnaire to the entire list, but rejected the idea on the grounds that sending a large email message which requests a response would annoy many subscribers. Since the intended audience for the questionnaire is MCS users, I decided to make the questionnaire available directly through the MCS archive. The questionnaire has been designed to only require two minutes to complete. It consists of a series of rating questions (e.g., "Overall, how would you rate your satisfaction with this new archive?" with answers ranging from 1 to 5). After the system was in

use by the mailing list's community for a few weeks, the questionnaire was advertised on the top page. Since the questionnaire is a web form, data collection and analysis was straightforward. The questionnaire itself can be found in Appendix D.

The questionnaire solicited data on both adoption and preference. Since it required effort to fill out and submit the questionnaire, it gave an accurate lower bound on how many people are using the archive. The questions regarding archive preferences provided direct data on whether users prefer MCS to traditional archives.

4.4 Study Implementation

The case study was implemented in several stages:

1. The MCS software was completed and an internal baseline release was made.
2. A trial run condensation was performed on an much smaller mailing list archive which consisted of roughly 165 messages. This allowed testing of the editing tool and the search facilities before exposing the system to a broader audience.
3. The MCS software was revised from the lessons learned in the trial run.
4. The jcvs archive was condensed over several weeks.
5. The archive was demoed to colleagues here in the ICS department who provided a variety of suggestions for improving the search interface.
6. The MCS software was revised in light of the suggestions.
7. The jcvs archive was announced to the mailing list on January 24, 2000 (see Appendix B for the text of the announcement email).
8. Users were able to use the condensed archive at their leisure.
9. On January 31, 2000, an update email was sent to the list which provided some example searches in an effort to make users more aware of the MCS archive.
10. On February 10, 2000, an online questionnaire was made available via the top page of the MCS archive. An announcement was made to the mailing list informing users of the questionnaire's existence and encouraging them to fill it out.
11. On February 23, 2000, I ceased collecting data from the questionnaire.

Chapter 5

Case Study Results

This chapter presents the results of the case study. In Section 5.1, I describe the data collected as I condensed two archives, including both qualitative observations and quantitative data on the process. In Section 5.2, I present the data collected from archive users or potential archive users, including qualitative data from the questionnaire and quantitative usage data.

5.1 Editing Results

As I listed in Section 4.4, two archives were condensed in preparation for the case study. The first archive was the icemail archive and it was quite small. I condensed it as a trial run of MCS. The archive was released to the subscribers of the list; however, there was almost no use of the archive by subscribers. I attribute this to the small size of the list (166 subscribers when the archive was condensed) and the very slow traffic (roughly three messages a week). Because there were no archive users, no data on the icemail archive is presented in Section 5.2. They are included in this section because the editing experiences did provide valuable insights for this research. Links to both archives can be found on the MCS research web page [2].

5.1.1 Editing Metrics

As I condensed the two archives, I recorded how much time I spent using the Leap toolkit [19]. I recorded time spent reading messages, condensing them, and any external reading required to condense the messages (e.g., a CVS reference book [20]). I did not record time spent fixing any critical defects in the MCS software as they were discovered as that time is not relevant to

Table 5.1. Editing time results for two condensed archives (all times in minutes)

Mailing List	Messages Examined	Condensing Time	Average Time Per Message
icemail	166	481	2.90
jcvs	1428	2165	1.52

determining the expected time required to condense future archives. Table 5.1 shows the time data for both archives.

As you can see, I took substantially less time per message examined when condensing the jcvs archive compared to the icemail archive. I attribute this to two factors: tool improvement, and editor improvement. The editing tool had a variety of quirks and defects when the first archive was condensed, so the condensation required substantial manual effort. After I condensed the first archive, I made many improvements to the editing tool which increased the speed with which messages could be condensed. In addition, I learned more about how to condense from my experience condensing the first archive. My increased knowledge allowed me to spend less time thinking about those issues when condensing the second archive. With more practice and enhancements to the editing tool, it should be entirely possible to bring the amount of time spent per message to one minute or lower. For a medium to low traffic mailing lists, this seems like an entirely acceptable amount of time to spend editing, particularly since this includes the time required to read the email for the first time, which an editor would presumably be doing anyway.

Table 5.2 shows a summary of the contents of the archives. As you can see, the archives contain only a fraction of the messages examined (23% for icemail, 12% for jcvs). This is to be expected because one of the goals of MCS is removing unnecessary messages. The jcvs list had a smaller percentage retained than the icemail list, presumably due to the heavier traffic of the jcvs list. For both lists, the number of keywords is fairly close to the number of archived messages. Because most messages contain multiple keywords, this indicates that many messages used the same keywords, otherwise the number of keywords would be larger than the number of archived messages. Symptoms were also fairly common on both lists: 21% of icemail problems had symptoms, 30% of jcvs problems had symptoms. The relatively high incidence of problems with symptoms indicates that the symptom search can be a useful search technique. To prove this conclusively, I would need to determine what percentage of symptoms actually encountered by users are actually included in the database.

Table 5.2. Statistics on the composition of two condensed archives

List	Msgs Examined	Msgs Archived	Problems	Keywords	Symptoms
icemail	166	39	19	40	4
jcvs	1428	177	82	120	26

5.1.2 Editing Experiences

While condensing the mailing lists, I kept a diary of thoughts and observations that occurred to me. In this section I will present the interesting parts of the diaries, and a process I developed to perform the condensation.

Many of the diary entries concern defects found in MCS or improvements that could be made to MCS. For example, when condensing the icemail list, there was a serious defect in MCS when saving a reloaded message: it would cause another message to be deleted as a side effect.

One improvement which became immediately obvious to me when the editing began was the order in which messages were displayed in ICEMail. ICEMail sorts messages by date from the most recent to the least recent. This makes sense for normal mail reading since you are usually most interested in recent messages. However, for editing purposes, it makes much more sense to display the messages from least recent to most recent so the editor can see problems before their proposed solutions. To make matters worse, when a message is deleted, the next message is automatically selected. This means that even if you start at the bottom of the list of messages (least recent), the selection moves back to older messages every time you delete one. I fixed both of these problems after I finished the icemail condensation, and before I condensed the second list.

The task of condensation boils down to understanding the content and relationships between messages. Understanding the content can be achieved straightforwardly by reading every message under consideration. This effort grows linearly with the size of the archive. However, understanding the relationships between messages grows as the square of the size of the archive. Therefore, my central concern when editing was figuring out how to consider the interrelationships of all the messages to be condensed simultaneously. Every message could potentially relate to any other message which makes the process difficult. By the time I had completed condensing the second list I had developed a multi-phase process for condensation to address this problem. The phases are:

1. Read every message in the archive, and delete those that are obviously not useful (reposts, advertisements, incoherent messages, etc.).

2. Scan through the remaining messages, and condense any problem-solution threads that are relatively short and appear only once in archive. After condensing the thread, delete it from the mailbox. Any thread that is long or occurs more than once should be moved to its own newly created mailbox bearing a name descriptive of the thread.

3. Scan through each of the thread mailboxes, condensing each thread one at a time. After condensing a thread, delete the mailbox.

4. By this point all the messages should be deleted.

These phases streamline the process of building a condensed archive from scratch using an existing full text archive as the source. The first phase allows the editor to read all the messages without having to make any decisions about condensation other than removing the totally useless messages. At the completion of the first phase the editor should have a feel for the archive. In the second stage the editor cherry-picks the easy problems for immediate condensation. The longer or repeating threads are deferred by placing them in individual mailboxes. The third stage is the hardest because these are the complicated threads, but at least the relevant messages are all in the same mailbox for easy perusal by the editor. By creating the individual mailboxes for each thread we have reduced the exponential complexity to something more manageable.

One of the more annoying problems for the editor in the current system is linking problems and solutions together. To link a problem to a solution, the editor places the message-ID of the solution message into the Links field of the problem message. However, the editor is condensing the thread sequentially so the solution message's message-ID is not immediately at hand since it hasn't been condensed yet! The workaround that I used was to copy the message-ID of the solution message to the clipboard before editing the problem message. Then the message-ID can be pasted into the Links field of the problem. The same thing can be done with the problem message's message-ID before condensing the solution message. This problem could be resolved by having some way in the user interface to indicate the problem-solution pair of messages being condensed. Then both messages could be linked together by the tool, instead of by hand.

As I condensed the archives, a variety of practical issues came up. One of the assumptions of the MCS implementation is that each message has a unique message-ID. This is a natural assumption because each email message from the mailing list should contain a unique message-ID according to Internet standard STD11 [21]. This assumption also ensures some correspondence between MCS messages and their origins. However, this assumption breaks down when a single message from the mailing list needs to be split into multiple messages in the archive (a message

containing multiple problems for example), or when a multiple messages from the mailing list need to be merged into a single message in the archive (such as the case when two messages each contain part of a single solution). I encountered this message fission problem most frequently when a user had written the author of the system a private email explaining the problem they were encountering, and the author had copied his reply to the public mailing list. To solve the problem of one list message splitting into multiple archive messages, I decided to use the original message-ID for the first part and generate a new message-ID for each additional part by appending ".MCS-*N*" (where *N* increments for each part) to the original message-ID. Each of the edited messages has the same unabridged message associated with it. Unfortunately, MCS requires that the message-ID of the edited and unabridged messages be the same, so there are multiple copies of the unabridged message in the unabridged database. This also means that the extra copies of the unabridged message have the altered message-ID of their edited counterpart. This is a regrettable side-effect because the unabridged messages are supposed to not be altered in any way. However, the minor and obvious changing of the message-ID in the relatively small number of messages which had to be split seems like an acceptable workaround to this problem. The problem of multiple list messages being merged into one archive message doesn't have any easy solutions in the current MCS architecture. One solution would be to change MCS so that one could specify authorship of text on a finer grain than a message, but this would require major changes to the message storage and user interface of MCS. The workarounds that I used were to either put the solutions in separate messages, or, if one of the solutions was minor, to place a paraphrase of it in editor notes of the main message.

Other issues I encountered while condensing involved judgment calls I had to make. One of the things I noticed was that it isn't always easy to decide whether or not a message should be included in the archive. These marginal messages included: problems or solutions which were poorly explained or confusing, problems that appear to be caused by "pilot error" or unique circumstances, or problems that have no solution. When faced with these kinds of messages I decided whether to include them or not based on my impression of their long-term usefulness: would anyone ever care about this message? Another area where I had to make judgment calls was the removal of irrelevant social aspects from messages. Many messages start with text like "Hello, my name is..." and end with *signature files* containing contact information. I decided to remove this kind of information from all messages since it is rarely relevant to archive users, but this could conceivably cause backlash from users. Users might like these social niceties and it's not clear that there is any substantial benefit from removing them. So far no user has complained about the removal of this kind of text.

The editing of message bodies brings up another potential issue for users. When I removed, changed, or added text to messages I always wrapped the changes in the editing tags discussed in Section 2.2.2, which ensures that users can differentiate between the original author's text and the editor's text. However, the tags do not differentiate between the different kinds of changes: a user cannot tell if text added summarizes something the original author said, or if the text is an addition written by the editor. The two kinds of changes are potentially very different since authors might be more concerned with editors adding text to their messages than they are by paraphrasing by an editor. Of course the user can always look at the unabridged message to do their own comparison. If this distinction turns out to be important to users, additional tags could be added to differentiate the different kinds of editing changes.

The building and maintaining of the keyword hierarchy, another major component of the editing task, provided additional challenges. Building the tree from scratch proved to be quite daunting because it required me to think not only about what keywords would be appropriate for a particular message, but also if the keyword would be relevant to other messages and whether this keyword fits under an existing category or not. This task increases the difficulty of condensing the first set of messages because of the overhead involved in defining and organizing the new keywords. Future editors might find it easier to start out using a keyword hierarchy from an existing (and hopefully related) MCS archive. I used the icemail hierarchy as the starting point for the jcvs archive. A common problem when creating keywords is what to call them. For example, if a message discusses the Java package Swing 1.1.1, what keywords should it have? If you simply use the keyword "Swing 1.1.1", then an archive user who is only interested in the differences between the Swing 1.0.X versions and the Swing 1.1.X versions may miss the message. What about users who are searching for Swing in general? The solution I used (primarily for keywords containing a version number) was to create a top level keyword ("Swing"), and then a general version number if appropriate ("Swing 1.1.X"), and then the actual version number being discussed if appropriate ("Swing 1.1.1"), with the versions grouped together under a category. This allows users to search across all those different levels of abstraction.

5.2 Archive User Results

In this section I detail the results obtained from archive users or potential users. First, I cover the information obtained through informal interviews and demos. Then, I cover the data collected in the web server log files. Finally, I present the results of the online questionnaire.

5.2.1 Interview Data

Before announcing the availability of the jcvs archive to the mailing list, I solicited feedback from members of my research group. I gave a demo of MCS at one of my group's weekly meetings. The demo elicited a variety of suggestions for improvements in the MCS interface. The original MCS archive web interface presented the user with a form containing all four of the search methods at once, which my colleagues pointed out was confusing. Based on their suggestions, I rewrote the search page so that the search methods were only displayed one at a time. They also suggested that I reduce the amount of text on the front page of the archive to simplify the experience for archive users. These suggestions led to my switching to a modal search form, because the original form was displayed on the front page and took up considerable screen real estate. After implementing their suggestions a few days after the demo, I showed them the interface again. They had additional comments and I went through several iterations before settling on the current interface.

In addition to the other members of my research group (who were already rather familiar with MCS from previous presentations and prototype demos), I asked one of the ICS faculty members, Edoardo Biagioni, to try out MCS briefly. Edo is one of the members of my thesis committee, so he was somewhat familiar with the MCS research but he had not seen the user interface. I asked him to perform a few searches and watched over his shoulder as he used the system. The first usability scenario I asked him to attempt was a symptom search. I had constructed a Windows NT batch file which launched the jCVS program with an appropriate CLASSPATH, except that I deliberately left out the "swingall.jar" file which contains the required Swing GUI libraries. Because it was lacking the Swing libraries, when Edo launched the batch file at my request, it failed with a stack trace similar to the one shown in Section 1.4.2. I asked him to attempt to find the solution to this problem in the condensed archive. He correctly ascertained that the symptom search would be the best search method in this case, and he was able to find the appropriate solution in the archive. I asked him to continue to browse through the archive, and he was able to select and initiate a keyword search without prompting. After the brief session was over, I asked him if the keyword selection interface reminded him of any other web site. I was expecting that he would say "Yahoo" or some other directory service, since that is what the interface was patterned after. Interestingly, he said that the interface reminded him of Dell Computer's online ordering site. Dell's online ordering system allows customers to customize their PC purchase by selecting from a range of computer components. These additional components are recorded in the interface until the customer is ready

to actually place the order for the computer. While the comparison is interesting, it may indicate that the keyword selector interface is too complicated, if it seems similar to an online ordering system.

5.2.2 Web Log Data

At the end of the questionnaire period, I copied the web server log file for analysis. The logfile contained data from January 22 to February 23. Using a program called *analog* [22], I analyzed the log file. The raw results of the analysis can be found in Appendix C.

Since the web logs were used to assess adoption of the MCS archive, it was important to screen out data that would skew the results. For this reason, I configured analog to ignore all requests which originated from either the "hawaii.edu" domain or the "lava.net" domain. I eliminated "hawaii.edu" because my testing and development was done from that domain, and I eliminated "lava.net" because my advisor's home access and my own home access are provided by that company. By eliminating the log entries from these two domains, all the internal usage of the archive should be removed from the data set.

According to the analysis by analog, the web server received requests from 99 distinct IP addresses. As mentioned in Section 4.3.3, this value is almost certainly an overestimate of the actual users since some users probably accessed the archive from different computers. However, this value is probably an upper bound on the number of users of the archive.

Analog also generated what it calls an organization report which uses the organizational analysis technique also described in 4.3.3. Analog has a table of domains and the number of levels required to identify an organization. Using this table, it attempts to group together all the requests from a particular organization. Therefore the organization report provides an estimate of the number of organizations have accessed the web server. Since every organization consists of at least one person, we can obtain a lower bound for the number of users by simply counting the number of entries in the organization report.

There were 70 entries in the organizational report. Some of the entries are not actual users such as the googlebot.com entry which is presumably a spider which collects data for the Google search engine [23]. The last nine entries have only one request which indicates that they didn't really do anything meaningful with the archive. On the other hand, however, there were 176 requests from IP addresses which could not be mapped into domain names which would presumably raise the organization count if they could have been resolved. The organization list also counts multiple users coming from the same organization as one, which could cause an underestimation of the number of

users. On balance, the value of 70 is probably a better estimate of the number of actual users than the 99 distinct IP addresses mentioned earlier.

Now that we have an estimate of the number of users of the archive, we can estimate the percentage of list subscribers that used the archive. As stated in Section 4.1, the list had approximately 401 subscribers at the start of the case study. Using the figure 70 as the estimate of the number of archive users, we find that this accounts for roughly 17% of the list membership. This exceeds the 16% goal which I set as the measure of whether or not the list subscribers had adopted the condensed archive.

There are a few other interesting insights available from the web log data. One involves a feature of Microsoft's Internet Explorer 5 for Windows product. Internet Explorer 5 allows web sites to provide custom icons to users who add the site to their list of favorite web sites (also known as bookmarks). When a user adds a site to their favorites list, Internet Explorer 5 will send an HTTP request for the file "favicon.ico". If it exists, this file is downloaded and used to provide a custom icon next to the item in the Favorites menu. While this is a minor improvement on the browser side, it provides additional information to the web site maintainer which is not normally available: an indication when a user has bookmarked their web site! By searching the log file for requests for this file, I found that it had been requested by five distinct IP addresses. Therefore, we can conclude that the archive was added as a bookmark on at least five different computers. This is only a lower bound since this feature currently only exists on Internet Explorer 5 for Windows.

By manually examining the log file I was also able to determine that the symptom search mechanism was being used incorrectly on a routine basis. Users were typing in symptoms such as "red files" or "can't add multiple files to current directory". Both of these examples appear to be natural language descriptions of symptoms, but they are not error messages provided by the jCVS program. For this reason, both these searches return no results. It appears that some users believed that the symptom search was some sort of natural language search which could work from a description of a symptom to the problem which causes it. I might be able to solve this confusion by adding additional explanatory text to the symptom page just above the text entry box. Another possible solution would be to change the name of this type of search from "symptom search" to "error message search", which is more descriptive.

5.2.3 Questionnaire Data

I made the questionnaire available on the archive web page from February 10-23. A total of six questionnaires were submitted. The questionnaire itself can be found in Appendix D, and the answers provided by users can be found in Appendix E.

I classified the six questionnaires returned into three different groups: those who had used neither the old archive nor MCS, those who had only used MCS, and those who had used both the old archive and the MCS archive. Each group had two questionnaire results which fit the characteristics. Due to the small number of questionnaires returned, I limit my analysis to qualitative trends that I noticed in the data.

First I will examine the results from each question. Included in brackets after each question is the total number of responses for that question (omitting responses of "Not applicable").

1. [6] Most of the respondents had substantial experience using the jCVS program. Except for one respondent who had never actually used jCVS, all respondents had used it for 3 months or more.

2. [6] All but one respondent said they were subscribed to the jcvs mailing list. It's not clear how the one respondent heard about the MCS archive without being subscribed to the mailing list. This respondent might have received a forwarded email about the archive.

3. [4] There was substantial variety in how frequently the respondents reported reading the mailing list messages. Some reported reading messages as soon as they arrived in their mailbox, while another said they almost never read the messages. It's not clear why one respondent who said in question 2 that they were subscribed answered "not applicable" to this question.

4. [5] There was also a lot of variety in how many messages the respondents reported reading from the list. Two users said they read every message, while two others said they read less than a third of the messages.

5. [5] Only two of the respondents reported actually having used the old, existing jcvs list archive. This may indicate that the old archive was not very well known to list users, or that the existing archive was not considered useful.

6. [2] The two respondents who had used the old archive had both used it on more than one occasion.

7. [2] The two users of the original archive both said that only sometimes would they they find what they had been looking for.

8. [4] This question about MCS usage frequency revealed rather light usage of the MCS archive by respondents. In fact three of the four respondents had only used the archive once at the time they answered the questionnaire.

9. [4] The respondents varied on how often they were able to find what they were looking for in the MCS archive. Two respondents said they always found what they were looking for, while one respondent reported having success only rarely.

10. [2] Both users who had used both the old and the MCS archive preferred the MCS archive to the old one.

11. [4] The four users who had used the MCS archive were split with two users reporting complete satisfaction with MCS, and the other two reporting only partial satisfaction.

12. [5] Respondents were divided on whether they would be willing to volunteer to be editors. Two said they were willing to be editors, two were unsure, and one was not willing.

The small number of responses was somewhat disappointing: two of the respondents hadn't even bothered to try the MCS archive before filling out the questionnaire (despite the paragraph before the questionnaire which asks that people use the archive before filling out the questionnaire), and only two respondents had used both the old archive and the MCS archive. However, the two users who did use both archives reported that: they always found what they were looking for in the MCS archive, they were completely satisfied with the MCS archive, and that they were willing to volunteer as editors. This makes some sense: in order to fully appreciate MCS you need to have used traditional mailing list archives. The willingness of respondents to consider volunteering to be editors is encouraging, and provides some hope that the burden of editing could be spread out among multiple editors.

The open-answer questions also provided some useful feedback (the full text of the answers can be found in Table E.2). The first open answer question provided respondents with the opportunity to suggest other mailing lists that would benefit from having a condensed MCS archive. Four of the six respondents indicated that they had mailing lists which they would like to see condensed. This indicates to me that there is interest in using MCS on other mailing lists. The general comments were quite varied but primarily upbeat and supportive of the MCS paradigm.

Overall, the questionnaire data seems to indicate that the MCS archive was useful to those who actually bothered to use it. While the sample size is very small, those who used both archives did indicate that they preferred the MCS archive to the existing one.

Chapter 6

Related Work

There are a variety of systems and research related to maintaining and searching collective memory. Here, I examine several such systems and compare them to MCS. Some of these systems are somewhat informal (like moderated mailing lists and FAQ files), and some are formal research projects. The informal systems are based on the author's knowledge of those systems and generally do not have references, because they evolved from common Internet practices.

6.1 Moderated Mailing Lists

Some mailing lists address the signal to noise problem by having a moderator or a group of moderators. All submissions to the list are forwarded to the moderator(s) who read the messages and decide whether or not to distribute them to the list. On most lists, the moderator(s) do not edit the messages submitted. They just choose whether or not to distribute the message. Also, to allay fears of censorship on the part of the subscribers, usually the criteria used to decide whether to distribute a message are rather liberal, e.g., the message is related to the topic of the mailing list and not an advertisement [24].

While moderation can be useful for maintaining a high signal to noise ratio, it suffers from several problems addressed in the design of MCS. Moderation requires a substantial commitment on the part of the moderator(s) to review submissions in a timely manner. Failure to do so halts all traffic on the mailing list and annoys subscribers who have come to expect the short turnaround time that digital media can provide. Moderators also tend to face continual concerns from subscribers as to whether they are moderating in a fair and consistent manner. Since the whole point of moderation is to prevent the distribution of inappropriate material, there is no way for a subscriber to tell whether

or not submissions are actually being judged by the stated criteria or whether the moderator(s) are acting on a whim or out of spite. Finally, moderation only partially improves the archives of a mailing list. Moderation will reduce the size of the archive and improve the average quality of a message in the archive compared to the archives of non-moderated lists. However, moderation does not solve retrieval problems, and due to the time pressures faced by moderators, they rarely have time to do more than a cursory check of submissions.

MCS reduces or eliminates all these problems with moderated mailing lists. One way of thinking about MCS is a form of moderation of the archives of a non-moderated list. Since MCS relates to the archive and not the list itself, the issue of timeliness is much less crucial: if you need to know what happened today on the list, you should be reading the list itself, not the archive. Also, since MCS does not affect the list distribution itself at all, most concerns about censorship should be eliminated. MCS provides a link from each edited message to the original unabridged message so users can easily see what was edited out or changed in any particular message. A truly suspicious user could even compare the MCS-condensed archive to other unabridged archives of the list since MCS archives are designed to exist in parallel with traditional archives. Finally, an MCS editor can remove or rewrite parts of a message long after the message is sent to the list when necessary to make the message more useful which is not done in a traditional archive even of a moderated list.

6.2 Description and Review of Mailing Lists

Robert C. Pedersen has done some preliminary work on the subject of describing Internet mailing lists [25]. His goal was to come up with a quantitative method for describing the content of a mailing list so that potential subscribers could make an informed decision on whether or not to subscribe. To do this, he devised nine categories for messages: administrative, announcements, discussion, information exchange, metadiscussion, networked resource pointers, noise, organizational communications, and position announcements. These categories were designed for use on mailing lists related to librarianship. He then subscribed to 13 librarian-related mailing lists, and over the course of 29 days, he classified all messages sent to the lists using the categories previously listed. With this data he was able to determine the average number of messages sent to the list per day, and the distribution of messages over the categories. He found that the distribution of message types was a good descriptor of the mailing list. While MCS has only two types of messages instead of nine, it does display statistics on the front page of the archive such as the number of messages in the archive, and the date of the last modification of the archive.

In a second article he recommends that mailing lists be reviewed in the same way that movies or books are to provide further assistance to potential subscribers [24]. Again, this would be a useful addition for MCS users who are potential list subscribers. While MCS need not provide any automated support for writing reviews, it makes sense for the MCS archive to provide a concise description of the mailing list being archived and what kinds of material a user would be likely to find within. This information can be provided on the front page of the MCS archive, or it could refer to an informational page elsewhere (possibly maintained by the mailing list administrator).

6.3 Frequently-Asked Question Files

Most frequently-asked question (FAQ) documents attempt to provide a similar service to MCS: a condensed version of important and useful information that came from a mailing list or newsgroup. There are several important differences between the two systems. FAQ files are usually maintained without specific tool support so they require extensive effort on the part of the maintainer to create and update. FAQ files are generally created with the intention of easy distribution either as plain text or HTML. Because of this requirement, FAQ files are mostly limited in size to a few hundred kilobytes and they are laid-out to be easy for humans to read. Since FAQs cannot be of arbitrary size and complexity, they must omit useful information.

MCS does not have these limitations. Since the system is not intended to be distributed by FTP or by posting to a mailing list or newsgroup, it can be as large as is necessary. A sophisticated query system is an integral part of MCS, so it is not necessary that the underlying data be structured in an easily understandable human format. Because MCS lacks these two restrictions, it need not limit the archives it creates to merely frequently-asked topics, it can contain any information that would be useful regardless of how broad its appeal.

The Internet FAQ Consortium [26] maintains an index of many FAQs and has some outlines of a plan to write a book on FAQs.

6.4 FAQ FINDER

FAQ FINDER allows users to quickly find answers to questions by searching a database made up of FAQ documents posted to Usenet [27]. The user enters his or her question into the system in natural language. First the system uses standard information retrieval techniques to determine which FAQs in the database are most likely to contain the answer to the question. It presents the top

five FAQs to the user, who can select the most likely candidate. Then the system uses a combination of lexical and semantic similarity checks between the asked question and the question-answer pairs in the FAQ file. It then presents the five most likely pairs for user consideration. A live version of the system can be found at the University of Chicago web site [28].

While FAQ FINDER is an interesting system, it is attempting to solve a different problem than MCS. FAQ FINDER assumes that there exists a large number of FAQ files which are already organized in question-answer format, and from those files it attempts to help users find the answer to their questions. The designers of FAQ FINDER explicitly chose not to implement any domain-specific knowledge into their system because their intended dataset is a large number of unrelated FAQ files. MCS attempts to create a FAQ-like body of knowledge from a mailing list, and then present the condensed information in useful, possibly domain-specific ways. In this way MCS attempts to solve the problem of getting the information into an FAQ-like state, which is already presupposed in FAQ FINDER. It might be possible to create a "stub" FAQ which FAQ FINDER could index, and if the user's question is a good match, FAQ FINDER would just send the user to the MCS-created archive.

6.5 Answer Garden

The Answer Garden system is designed to provide an "organically" growing database of answers to questions by end-users [29]. Users interact with the system by answering a series of diagnostic multiple-choice questions which lead them through the tree of answers already in the system. If users find that their questions are not answered in the database, they can enter their questions into the system and it will be forwarded to an appropriate expert via email. When the expert answers, the result is sent back to the original question-poser and also inserted into the tree for future retrieval.

Answer Garden's goal in life is to answer questions. Like MCS, it uses human input to decide what questions and answers should be in the database. However, Answer Garden is really only suited to the task of answering questions. A user who just wants to browse information either has to answer the diagnostic questions or guess where on the tree the information might be located. It also requires a group of experts to be responsible for answering the questions posed by users. In an organization where certain people's job function is answering the questions of others in the same organization, this works well because users get answers efficiently and experts don't have to answer the same questions over and over. However, the assumption that there is a pool of experts who are

required to answer questions falls down in a volunteer user community where nobody is required to do anything. In MCS, experts can answer questions posed to the list at their whim; only the editor is required to work in order to keep the system functional. MCS also does not require users to use any special software to continue participation in the mailing list, while Answer Garden assumes that all users will use the Answer Garden tool when they have a question. In addition, MCS provides the symptom search method which allows a user to use an error message to find the solution to a problem immediately. Answer Garden requires users to answer a series of diagnostic questions, with no way to short circuit the process.

Finally, the information in Answer Garden only grows as the system is used, while the information in MCS grows as long as there is useful traffic on the mailing list.

6.6 Answer Garden 2

Answer Garden 2 is a refinement of the Answer Garden system in Section 6.5. It improves on Answer Garden by adding a system of gradual escalation for questions input into the system (thereby providing more context to the person answering the question), and a subsystem for collaboratively "refining" the information in the database [30]. All of this is built on a set of versatile and configurable components which allow the system to be tuned for a particular environment.

This system appears to implement many of the features required for MCS. The system which inputs data into the system (CafeCK) provides a mechanism for capturing mailing list messages, and the "refining" system called Co-Refinery allows collecting, culling, organizing, and distilling information. The Co-Refinery system seems particularly close to MCS's requirements. Unfortunately, Answer Garden 2 is not available for public distribution [Ackerman, personal communication], so the actual implementation was not available for use as a foundation for MCS. In addition, there does not appear to have been an evaluation of the Answer Garden 2 system in the field, so it is difficult to obtain further insight into the differences between MCS and Answer Garden 2.

6.7 Faq-O-Matic

Faq-O-Matic was created to solve some of the same problems MCS addresses: the difficulty in finding answers in mailing list archives, and the substantial effort required to maintain an FAQ. Faq-O-Matic addresses these issues by creating a dynamic WWW-based FAQ which any

member of a user community can contribute to. Any user can browse through the web pages and make additions as necessary. This provides an easy way to maintain an FAQ since any member of the community can volunteer to help. The system offers limited access control, but this must be balanced against the need for openness since contributions are from volunteers. There is a provision for moderation of the FAQ, and moderators can move or delete contributions. However, there is no centralized authority in charge of the FAQ, so pieces of potentially incorrect or mutually conflicting information can be posted. Furthermore, new additions have to written from scratch by contributors, unlike MCS. Simple keyword searching of the FAQ is provided. Documentation on Faq-O-Matic is provided through an FAQ maintained using Faq-O-Matic [31].

6.8 Open Directory Project

The Open Directory Project (ODP) [32] is a large directory of web sites categorized by human editors. Like Yahoo! [14], it attempts to list the best web sites relevant to every imaginable subject area. However, unlike Yahoo!, there are 22000 editors who are all volunteers. An expert in a particular subject area can register at the site to become an editor. Editors look for the best web sites in their subject area and add them to the directory. Since the web sites are selected by human experts, the quality of the links can be higher than those generated through automated techniques. Users can browse the categories or perform keyword searches.

The ODP addresses a different problem than MCS, but it uses human effort to remove low-quality information in the same way as MCS. It supports massively parallel editing of the directory, while MCS currently supports only a single editor. However, it does not support any type of domain-specific searching as it attempts to support all types of web sites.

6.9 Slashdot

Slashdot is a popular news and discussion web site which describes itself as “News for Nerds” [33]. Slashdot runs on a system called Slash (which stands for Slashdot-Like Automatic Storytelling Homepage) which was written specifically for Slashdot. There are several human editors who select which news stories to present to readers. The stories range in length from a few sentences to several pages. After each story, users can add comments and discussion in a threaded format, either anonymously or with attribution. Each message has a point value associated with it, ranging from -1 to 5, where higher point values should indicate more useful or interesting

messages. Anonymous messages start at value 0, and attributed messages usually start at value 1. Registered readers of Slashdot can set a default threshold, and all comments below that threshold will not be displayed. This configurability allows readers to choose the quantity and, hopefully, the quality of comments they read.

Messages gain or lose points through a process of moderation. Moderators are selected from the pool of registered readers based on a complex set of criteria including the frequency of reading Slashdot (favoring regular readers), and the amount of time since initial registration as a reader (favoring long time readers). When selected, a moderator can annotate comments with descriptive words such as “flamebait”, “informative”, or “redundant” chosen from a short fixed list. Positive words such as informative cause the comment to gain one point value, while negative words such as redundant cause the comment to drop one point value. The moderator status only lasts for a few days, and the moderator is only allowed to annotate a small number of messages during their session. Moderators are further restricted from contributing their own written comments to any story if they have moderated any comments on that story. The substantial restrictions on moderation were designed to spread the task of moderation out over the user community, and to ensure that no moderator could exert undue influence on the comments.

Slashdot also uses the moderation values to calculate a *Karma* value for each registered user. The Karma value is the sum of all moderations made to a user’s comments. Therefore, a user whose comments have usually been positively moderated will have a positive Karma and vice versa. Users with high Karma levels are given the option of adding a bonus point to any comment that they post, thus starting their comment with a point value of 2 instead of 1. There is even a system of meta-moderation which any user can participate in. When meta-moderating, 10 randomly selected comments which have been moderated are displayed, and the user is asked to vote on whether the moderation was appropriate. The results of the meta-moderation are factored into the moderator’s Karma value as an additional check against the abuse of moderator power.

Slashdot exists for a different purpose than MCS. Slashdot has created a community of users who are interested in reading and discussing the latest technology news, while MCS focuses solving problems rather than news dissemination. The extensive moderation facilities were created to deal with the problem of too many comments, which is similar to the impetus for MCS. However, Slashdot provides only keyword searching which limits the long-term value of the stories and comments. The Slashdot moderation facility only allows moderators to provide meta-level information to comments, unlike MCS where the editor can actually change the message body itself. Slashdot’s

moderation facilities are interesting, and providing something similar in MCS would be one way to include archive user feedback into the system.

6.10 Expertise Web Sites

A variety of companies have started web sites that attempt to match users with questions to experts who can answer them. Given their large number, I have written a detailed description of one such web system. Then, I provide a brief summary of some of the other available systems.

6.10.1 Experts Exchange

Experts Exchange is a virtual community, where users can ask questions which are answered by volunteer experts [34]. In order to encourage experts to answer questions, each question is assigned a point value by the author of the question. After the question has been posed, experts can propose answers or make comments on the question for further clarification. Once an expert has proposed an answer, the author of the question decides if the answer is satisfactory or not. If the author of the question judges the answer satisfactory, the he or she assigns a grade to the quality of the answer. Then the value of the question is subtracted from the author's *question point* total, and that value is multiplied by the grade and added to the expert's *expert point* total. This exchange creates an information economy where questions are assigned point values by authors which are proportional to the question's difficulty, and experts compete for the right to answer questions.

Users must register before they can pose questions on the web site. By registering, users receive 75 question points, and then receive 5 additional points every day they remain an active user. Expert points are not convertible into question points, but they provide recognition to valuable experts, and might be redeemable for prizes in the future. The system maintains grading histories for both normal users and experts, so that authors of questions who grade unreasonably and experts who provide poor answers can be avoided by other users.

Registered users can search through archives of Previously Asked Questions (PAQs). If a user finds a question which is relevant, they can view the question for free, but they must spend question points equal to 10% of the original price of the question to see the answer.

The Experts Exchange, like MCS, exists to solve users' problems in a quick and easy manner. Unlike MCS, it does not get its content from an existing data stream: it encourages the generation of the content directly through the point system. The Experts Exchange also encompasses both the creation of content, and the archival of that content for future retrieval, whereas

MCS is purely an archival system. The point system is an interesting technique for motivating the experts to participate. Mailing lists usually do not have an explicit motivator: users participate out of a desire to help or be helped. MCS relies on this implicit motivation or mailing lists rather than a currency system. Requiring users to spend points to retrieve answers from the archive appears to be a disadvantage for Experts Exchange, since it might encourage lazy users to pose questions which have been answered before.

6.10.2 Other Expertise Sites

There are a number of other expertise web sites, and the number seems to increase daily. Some of the systems like KnowPost [35], use a credit system like Experts Exchange, but simply charge one credit for asking a question and give one credit for answering a question. No differentiation is made between hard and easy questions. Due to the simpler credit system, KnowPost allows free access to answered questions.

Other systems such as infomarco.com [36], HotDispatch [37], InfoRocket [38], and EXP.com [39] use actual money as their currency. Since real money is being exchanged, users may be reluctant to ask questions. Using money as the currency also causes problems for archives, since this means that the system must charge users for access to the archive, or risk having users browse the archives for free instead of paying to get their question answered. Some of the systems don't have an archive for this reason. Others plan to add archives in the future, but charge users for viewing answers and pay the owner of the information (who could be the question poser or the expert that answered) a royalty each time the information is accessed. Some systems like Expert-Central [40] and QuestionExchange [41] attempt to solve the problem by having a hybrid payment system where answers can either have a price assigned to them, or can be free. Both systems have archives which contain all the free questions and answers.

Ithority [42] also uses money as the currency, but acts solely as a broker between clients and experts. All the information transactions take place outside of the web site.

Two other sites, ExpertCity [43] and NoWonder [44], focus on computer support. They are different from the other systems because they provide live experts for real-time help. Both sites provide software for screen sharing, so that the experts can directly view and manipulate the user's computer system.

All of the systems provide some sort of rating system for experts, and many provide a rating system for the users. These rating systems attempt to provide a way for users and experts to decide whether or not to enter into a transaction. While this kind of rating system could be added

to MCS, the editor in MCS implicitly rates messages by choosing which messages to keep in the archive. Since the systems here provide a currency exchange (often with actual money) users and experts want assurances that they will not be cheated. There is no analog to this problem in MCS since there is no explicit currency other than respect and goodwill.

6.11 The Coordinator

The Coordinator is a communication tool based on the language/action perspective, which views language as a means for directing the actions of oneself and others [45]. The Coordinator attempts to enhance human collaboration by explicitly supporting "conversations" such as requests and offers [46]. The conversation is viewed as having various states, and actions of the requester or the requestee can move between the various states. For example, a user who would like to have a paper reviewed, sends a request message to the reviewer with a deadline for reply to the request and a deadline for completion of the review. The reviewer can then select one of a finite list of options: accept the request, make a counter proposal, or decline the request. The idea behind this structure is that it allows the system to show the state of the conversations a user is having with other users. Coordinator can provide explicit reminders about commitments made, and ensure that there is no misunderstanding about what was requested, and whether the request was accepted.

Some have brought up problems with respect to the Coordinator. Lucy Suchman claims that the Coordinator and the language/action perspective enforce a certain worldview about how people ought to go about collaborating [47]. A survey of groupware across 25 organizations and 223 people found that many people ignored the speech act capabilities of the Coordinator and simply used it as an email program (specifying all messages as requests whether they were or not) [48]. Another Coordinator trial found that the benefits of the product were not offset by the effort required to use it [49].

In some sense MCS can be thought of as performing the opposite task as the Coordinator: extracting structure and meaning from unstructured dialog, as opposed to requiring users to specify the structure with the initial message. This reversal is crucial because MCS is designed to work with existing mailing lists made up of voluntary subscribers. If subscribers were forced to add structure and keywords when submitting new messages, or required to use a special software program to read messages, they would flame the person imposing this system to a crisp and abandon the mailing list *en masse*. Since MCS wants its interaction with the host mailing list to be as painless as possible, it must reverse-engineer the structure after the fact. If MCS becomes popular, it may be that some

advanced users will want to include MCS structure in their submissions to the list. I could support user-added structure by creating an authoring tool which is a subset of the editing tool. However, it is unlikely that these pre-structured messages would ever account for more than a fraction of the actual list traffic for the reasons cited in the paragraph above.

Chapter 7

Conclusion

The goals of this research were three-fold. First, to introduce the ideas of human involvement and domain-specific representations to the area of mailing list archives. Second, to develop a system for improving mailing list archives which demonstrates these ideas. Third, to test these ideas by actually constructing a condensed mailing list archive, and by collecting data to see whether the subscribers prefer it to existing archives.

This chapter first summarizes how these goals were achieved in this research, and presents the major contributions of this research. Then it discusses the future directions of this research.

7.1 Research Summary and Contributions

This section discusses the three major contributions of this research: new ideas for improving mailing list archives, the MCS system, and the case study.

7.1.1 New Ideas for Improving Mailing List Archives

My personal experience using the archives of product support mailing lists to solve problems led me to believe that mailing list archives could be improved. The two main improvements that I came up with are:

1. The involvement of human editors to improve the quality of the contents of the archives.

2. A focus on data representations and search methods which were domain specific.

By adding some human effort on the part of an editor, the messages in the archive could be categorized, linked, and trimmed to their essentials. Obviously this involves a trade-off since

human effort is relatively scarce compared to most computer effort, but the result could be much reduced effort on the part of the many archive users. The human editing factor also allows for the use of domain specific representations such as assigning a type to messages and opening the door to new search methods like the symptom search. Most search facilities attempt to be maximally generic in order to be applicable to the widest possible set of data sources, while MCS harnesses the domain specificity to make a particular kind of archive easier to use and more powerful.

7.1.2 Mailinglist Condensation System (MCS)

Another goal of this research was to implement some of the ideas for improving mailing list archives. In light of the time constraints of masters-level research, I chose to implement a system designed for improving the archives of product support mailing lists. Since these lists consist primarily of users describing their problems and other users explaining the solution to those problems, MCS focuses on exclusively on this aspect. All messages kept in the archive are either problems or solutions. An editing tool allows the editor to condense the messages and annotate them with meta-level information. This meta-level information is then used by the archive's searching subsystem to enable the four different search methods: keyword, symptom, 2D, and full text.

7.1.3 Case Study

The third goal of this research was to actually use MCS to condense an appropriate mailing list and find out whether users actually preferred it to conventional mailing list archives. I condensed over 1400 messages from two mailing lists to test whether MCS's editing features would actually work. I successfully created the archives at an average rate of 1.5 minutes per message, thus demonstrating that condensation is feasible. The archives were then made available to the lists' subscribers. On the jcvs mailing list the web log entries suggest that approximately 17% of the list subscribers used the MCS archive, exceeding the target threshold for adoption of 16%. There are also some limited indications from the user questionnaire that users prefer the new MCS archive to the conventional one.

7.2 Future Directions

With the completion of this phase of the MCS research, there are a variety of avenues for future work. In Section 7.2.1, I discuss some of the obvious improvements that could be made to MCS. In Section 7.2.2, I deal with the topic of recruiting one or more editors to maintain the

archives built for the case study. In Section 7.2.3, I explain the plan to make MCS freely available. In Section 7.2.4, I describe how to encourage other lists to adopt MCS for their archives.

7.2.1 MCS Improvements

I accumulated a variety of ideas for future improvements in MCS over the course of the research which could not be implemented due to time constraints. One of the most obvious is extending MCS so that multiple people can work collaboratively as editors. Supporting multiple editors will require a locking mechanism for both individual messages and the keyword tree to prevent one editor from overwriting another's work. Implementing locking for messages would be straightforward, but locking the keyword tree is more difficult. Access to the keyword tree is required by all editors throughout their session, so in the current design the editor tool downloads the tree at startup. A simple approach would be to have a mutex on the keyword tree so only one editor can have read/write access to it at a time. However this reduces the utility of allowing multiple editors since almost any editing will require adding new keywords to the tree. A more complicated solution would allow only one editor full read-write access and allow all other editors read-append access. The editors with read-append access could add new keywords, but not change existing keywords. This would require that the read-append clients remember the added keywords and report to the KeywordServer only those keywords to be spliced into the tree. And, of course, they can't be spliced into the tree until the read/write editor checks the tree back in. In addition, once there is a mutex of some sort timeouts will be needed or some way for an editor to break the mutex in case of editor negligence or an editor computer crash.

Another area which needs improvement is the editing tool. It needs to be even more efficient than it currently is. Many keystrokes and mouse clicks could be reduced by minor user interface improvements such as default values or selections. Improvements in editing efficiency will reduce overhead, and thereby make it more likely that other mailing lists will adopt MCS for their archives. In the message editor window, keywords could be displayed in a scrolling list or a pop-up menu rather than the current comma-separated text field, to make it easier to add or delete keywords.

From the archive user perspective, one concern that came up repeatedly was the complexity of the searching interface. This is important since most users will be expecting the canonical search interface which consists of a text field where they can enter arbitrary keywords. Because MCS uses a hand-picked set of keywords, the interface for the simple task of selecting a keyword was overly complicated. One reason for complication in the keyword selector is the ability to select

more than one keyword for a search. There are several possible solutions to this problem. If the ability to select multiple keywords was eliminated (or at least deemphasized), the keyword selector could be made much easier to use: selecting a keyword immediately initiates a search using that keyword. Another possibility would be to allow users to search through the keyword tree itself instead of having to browse it. However, this additional layer of searching could potentially add confusion. For archives with a reasonable number of keywords, one possibility would be to display all the keywords on a single web page in alphabetical order. Another improvement suggested for the archive would be to allow users to selectively expand regions of text which have been changed by the editor, instead of having to switch between the edited message and the unabridged version.

In the current implementation of MCS, the Java servlets generate substantial amounts of HTML. This means that even simple changes to the desired layout require the entire system to be recompiled, which is clearly suboptimal. A variety of systems exist for separating the HTML layout from the underlying implementation, such as JavaServer Pages from Sun [50], or WebMacro from the shimari project [51]. In addition to cleaning up the servlet code, this would make it easier for MCS to be adapted to the needs of other mailing lists.

Several aspects of MCS could be improved by moving from the current ad-hoc database system to a production-quality database system. While the current system has worked well for the case study, it is clear that it would not scale well for much larger condensed archives. Using the locking and transaction features of a production database system would make it much easier to support multiple editors. Finally, a variety of small problem would be fixed by moving to a real database, such as the current lack of consistency checking on the links between messages.

Since some users may still wish to have an FAQ for a mailing list, it would be possible to write an FAQ generator module for MCS. It would take all the problems and solutions in the database and format them as one large document. The linking of problems and solutions could be collapsed so that solutions follow the problems they solve in the document, and solutions that apply to more than one problem could either be repeated, or cross-referenced.

7.2.2 Editor Recruitment

The editor obviously plays a crucial role in the operation of MCS. Without continual updating, the database becomes of only historical interest. For the case study, I acted as the sole editor. To ensure the continued survival of the archive created in the case study, it will be necessary to recruit other editors. Given the responses on the questionnaire described in Section 5.2.3, it should be possible to get volunteers from the list to step forward as editors.

7.2.3 Open Source Distribution

As part of the growing Open Source movement [52], I would like to see MCS released in source form to the public. Easily downloadable source (and binaries for that matter) will encourage others to adopt the system for their mailing lists, and spur other researchers to build on the MCS framework.

7.2.4 Adoption by Other Mailing Lists

Convincing other mailing lists to use the software for their archives would be the final stage in moving the software out into general use. This adoption process may be more difficult because it requires the mailing list's community to embrace the system as well as recruitment of one or more editors from the mailing list.

Appendix A

“jcvs” Mailing List Subject Lines

In this appendix I present some actual Subject lines extracted from a few days of traffic on the jCVS mailing list. I provide this to demonstrate the chatty an disorganized nature of most mailing lists.

```
Subject: Re: [jcvs] HELP--authentication error
Subject: Re: [jcvs] Examples of Implementation
Subject: Re: [jcvs] failed authentication with the user name
Subject: [jcvs] Analog to cvs update -n -q?
Subject: Re: [jcvs] Analog to cvs update -n -q?
Subject: Re: [jcvs] Analog to cvs update -n -q?
Subject: Re: [jcvs] Analog to cvs update -n -q?
Subject: Re: [jcvs] Analog to cvs update -n -q?
Subject: Re: [jcvs] Analog to cvs update -n -q?
Subject: [jcvs] connect doesn’t seem to work
Subject: Re: [jcvs] connect doesn’t seem to work
Subject: [jcvs] admin -m / change log entries
Subject: [jcvs] cannont chdir(/root.home)
Subject: Re: [jcvs] Analog to cvs update -n -q?
Subject: [jcvs] License question..
Subject: [jcvs] Browsing projects with jCVSlet
Subject: [jcvs] Direct Connection with Kerberos
Subject: Re: [jcvs] admin -m / change log entries
Subject: [jcvs] Where WinInstaller
Subject: [jcvs] JCVS bug
Subject: Re: [jcvs] JCVS bug
```

Subject: Re: [jcvs] Where WinInstaller

Subject: [jcvs] kinda stuck...

Subject: RE: [jcvs] kinda stuck...

Subject: Re: [jcvs] cannont chdir(/root.home)

Subject: Re: [jcvs] Browsing projects with jCVSlet

Subject: Re: [jcvs] Direct Connection with Kerberos

Subject: [jcvs] Installation woes or ClassNotFoundException

Subject: Re: [jcvs] Installation woes or ClassNotFoundException

Appendix B

Introductory Email to jCVS List

The following email was sent to the jCVS list as an introduction to the existence of the condensed jCVS list archive:

```
Date: Mon, 24 Jan 2000 17:18:34 -1000
From: Robert Brewer <rbrewer@lava.net>
To: jcvs@mail.gjt.org
Subject: Problem Solving jCVS archive
Message-ID: <3841508885.948734314@sabrina.ics.Hawaii.Edu>
X-Mailer: Mulberry (Win32) [1.4.5, s/n U-300878]
MIME-Version: 1.0
Content-Type: text/plain; charset=us-ascii
Content-Transfer-Encoding: 7bit
Content-Disposition: inline

Mahalo for that kind introduction Tim!

As Tim mentioned, I'm working on my Masters degree in ICS here at the
University of Hawaii at Manoa.

My research is in the area of improving the archives of product support
mailing lists like this one. My basic thesis is that when people go to an
archive of a product support mailing list, it is usually because they are
having some sort of problem with the product and they want to find a
solution. Therefore the archive should be designed to maximize the
efficiency with which users can find solutions to their problems.

I have created a system called MCS (Mailinglist Condensation System) which
```

takes existing mailing list archives and turns them into problem solving archives. This is done through a process called condensation which takes the verbose content of a mailing list and removes the messages which don't have long-term relevance. The condensation is done by a human editor (for the jCVS archive this was me) who leaves out all the messages which are irrelevant to problem solving. The editor annotates the messages with keywords, writes a one-line summary of each message, and even removes extraneous text from the body of messages. The result is an archive which is much smaller with a higher information density and four methods by which searches can be performed: keyword search, symptom search, 2D search, and full-text search.

I hope you'll take the time to try out the archive and maybe it will save you some time by tracking down a problem you've been having with jCVS. In about two weeks I will be putting up an online survey at the MCS archive to gauge people's reaction to the system. Feel free to send comments about the system using the Feedback page.

The MCS archive is located at <http://csdl.ics.hawaii.edu:8100/> and Tim has graciously added it to the footer of each message to the list for easy access. I'll be continuing to condense new messages and add them to the archive.

Thanks for your time, and I hope the archive can become a valuable resource for the list which lets us spend more time discussing new issues and less time answering frequently asked questions. :)

Appendix C

Raw Web Server Log Analysis

This is the report generated by version 4.03 of the *analog* log file analysis program [22] when supplied with the log file from the web server running the condensed archive of the jcvs mailing list.

```
Web Server Statistics for jcvs Condensed Mailing List Archive
============================================================

Program started at Wed-23-Feb-2000 17:46.
Analysed requests from Sat-22-Jan-2000 08:57 to Wed-23-Feb-2000 17:18 (32.3
  days).
------------------------------------------------------------------------------

General Summary
---------------
(Figures in parentheses refer to the 7 days to 23-Feb-2000 17:46).
Successful requests: 949 (281)
Average successful requests per day: 29 (40)
Successful requests for pages: 240 (76)
Average successful requests for pages per day: 7 (10)
Failed requests: 11 (0)
Distinct files requested: 339 (135)
Distinct hosts served: 99 (31)
Unwanted logfile entries: 278
Data transferred: 2.626 Mbytes (837.015 kbytes)
Average data transferred per day: 83.159 kbytes (119.573 kbytes)
------------------------------------------------------------------------------
```

```
Monthly Report
--------------
Each unit (+) represents 4 requests for pages or part thereof.

   month: reqs: pages:
--------: ----: -----:
Jan 2000:  262:    63: ++++++++++++++++
Feb 2000:  687:   177: +++++++++++++++++++++++++++++++++++++++++++++++

Busiest month: Feb 2000 (177 requests for pages).
-------------------------------------------------------------------------------

Daily Summary
-------------
Each unit (+) represents 2 requests for pages or part thereof.

day: reqs: pages:
---: ----: -----:
Sun:   68:    30: +++++++++++++++
Mon:  167:    37: ++++++++++++++++++
Tue:  150:    32: ++++++++++++++++
Wed:  336:    67: +++++++++++++++++++++++++++++++++
Thu:   67:    23: ++++++++++++
Fri:  110:    38: ++++++++++++++++++
Sat:   51:    13: +++++++
-------------------------------------------------------------------------------

Hourly Summary
--------------
Each unit (+) represents 1 request for a page.

hr: reqs: pages:
--: ----: -----:
 0:   12:     2: ++
 1:   48:     8: ++++++++
 2:    7:     3: +++
 3:   40:     4: ++++
 4:   68:    28: ++++++++++++++++++++++++++
 5:    0:     0:
 6:   80:    18: ++++++++++++++++++
 7:   50:    14: ++++++++++++++
```

```
 8:  146:    11: ++++++++++
 9:   73:    13: +++++++++++++
10:   14:     5: +++++
11:   46:    16: ++++++++++++++++
12:   34:    10: ++++++++++
13:   47:    17: +++++++++++++++++
14:   33:     9: +++++++++
15:    0:     0:
16:   48:    16: ++++++++++++++++
17:   50:    16: ++++++++++++++++
18:    4:     4: ++++
19:    0:     0:
20:    8:     7: +++++++
21:   82:    22: ++++++++++++++++++++
22:   43:     9: +++++++++
23:   16:     8: ++++++++
--------------------------------------------------------------------------------

Domain Report
-------------
Listing domains, sorted by the amount of traffic.

reqs: %bytes: domain
----: ------: ------
 360: 37.08%: .com (Commercial)
 205: 21.56%: .net (Network)
 176: 18.77%: [unresolved numerical addresses]
  85: 10.37%: .de (Germany)
  26:  2.91%: .fr (France)
  25:  1.79%: .gov (USA Government)
  12:  1.39%: .dk (Denmark)
  12:  1.19%: .uk (United Kingdom)
  13:  1.15%: .au (Australia)
  12:  1.04%: .at (Austria)
   4:  0.66%: .edu (USA Educational)
   5:  0.65%: .se (Sweden)
   4:  0.41%: .fi (Finland)
   3:  0.29%: .br (Brazil)
   3:  0.27%: .arpa (Old style Arpanet)
   2:  0.23%: .be (Belgium)
   1:  0.12%: .mil (USA Military)
```

```
   1:  0.11%: .za (South Africa)
-------------------------------------------------------------------------------

Organisation Report
-------------------
Listing organisations, sorted by the number of requests.

reqs: %bytes: organisation
----: ------: ------------
 176: 18.77%: [unresolved numerical addresses]
  70:  6.81%: moorebcs.com
  39:  4.35%: collab.net
  35:  3.74%: uu.net
  33:  3.63%: eumetsat.de
  33:  3.31%: ses-astra.com
  28:  2.80%: earthlink.net
  28:  2.53%: pacbell.net
  25:  2.66%: nucleus.com
  24:  2.08%: best.com
  24:  2.87%: pacoffee.com
  24:  1.68%: lanl.gov
  24:  2.64%: ara.com
  22:  3.14%: dresdnerbank.de
  21:  2.27%: novell.com
  17:  1.61%: mindspring.net
  17:  1.50%: trustice.com
  14:  1.56%: ihost.com
  13:  1.15%: monash.edu.au
  13:  1.77%: multipath.com
  13:  1.06%: digex.com
  13:  1.64%: rmc.de
  13:  1.56%: cnc.net
  12:  1.19%: demon.co.uk
  12:  1.30%: mp3.com
  12:  1.59%: ubs.com
  12:  1.04%: tuwien.ac.at
   9:  0.92%: fedex.com
   9:  1.02%: fast-search.net
   9:  1.12%: fzk.de
   9:  1.03%: prserv.net
   8:  1.01%: kia.dk
```

```
   8:  0.95%: dialups.net
   7:  0.68%: codiciel.fr
   7:  0.54%: home.com
   7:  0.70%: hp.com
   7:  0.58%: kuit.com
   6:  0.27%: pncbank.com
   6:  0.94%: betasys.com
   6:  0.86%: silicomp.fr
   5:  0.45%: snap.com
   5:  0.65%: volvo.se
   5:  0.68%: univ-angers.fr
   4:  0.49%: bellsouth.net
   4:  0.21%: emn.fr
   4:  0.47%: uhc.com
   4:  0.41%: regex.fi
   4:  0.45%: uni-sb.de
   4:  0.37%: mediaone.net
   4:  0.38%: bfc.dk
   3:  0.37%: mich.net
   3:  0.33%: voyager.net
   3:  0.36%: univ-nantes.fr
   3:  0.33%: cnet.com
   3:  0.29%: acessonet.com.br
   3:  0.16%: cyrus.net
   3:  0.28%: rwth-aachen.de
   3:  0.27%: arpa
   2:  0.33%: msus.edu
   2:  0.23%: skynet.be
   2:  0.23%: eu.net
   2:  0.33%: rpi.edu
   1:  0.12%: af.mil
   1:  0.12%: capgemini.fr
   1:  0.12%: nasa.gov
   1:  0.11%: mweb.co.za
   1:  0.12%: digital.de
   1:  0.12%: verity.com
   1:  0.12%: brixnet.com
   1:  0.11%: lanxtra.com
   1:  0.11%: googlebot.com
-------------------------------------------------------------------------------
```

```
Directory Report
----------------
Listing directories with at least 0.01% of the traffic, sorted by the amount
  of traffic.

reqs: %bytes: directory
----: ------: ---------
 707: 70.26%: /servlet/
 215: 26.69%: [root directory]
  27:  3.06%: /help/
-------------------------------------------------------------------------------

File Type Report
----------------
Listing extensions with at least 0.1% of the traffic, sorted by the amount
  of traffic.

reqs: %bytes: extension
----: ------: ---------
 707: 70.26%: [no extension]
 167: 19.04%: [directories]
  73: 10.69%: .html [Hypertext Markup Language]
   2:  0.02%: [not listed: 1 extension]
-------------------------------------------------------------------------------

File Size Report
----------------

      size: reqs: %bytes:
----------: ----: ------:
         0:    6:       :
  1b-  10b:    0:       :
 11b- 100b:    0:       :
101b-  1kb:   14:  0.29%:
 1kb- 10kb:  925: 97.99%:
10kb-100kb:    4:  1.71%:
-------------------------------------------------------------------------------

Status Code Report
------------------
Listing status codes, sorted numerically.
```

```
reqs: status code
----: -----------
 948: 200 OK
   1: 304 Not modified since last retrieval
  11: 404 Document not found
-------------------------------------------------------------------------------

Request Report
--------------
Listing files with at least 20 requests, sorted by the number of requests.

reqs: %bytes:       last date: file
----: ------: ---------------: ----
 305: 23.94%: 23/Feb/00 17:17: /servlet/MCSSearch
  53:  2.60%: 23/Feb/00 17:16:   /servlet/MCSSearch?mode=keyword
  37:  1.77%: 23/Feb/00 11:23:   /servlet/MCSSearch?mode=twod
  28:  1.68%: 23/Feb/00 11:23:   /servlet/MCSSearch?mode=symptom
  27:  1.52%: 23/Feb/00 09:55:   /servlet/MCSSearch?mode=fulltext
 298: 38.48%: 23/Feb/00 17:17: /servlet/MCSKeywordSelector
  38:  4.32%: 23/Feb/00 17:16:   /servlet/MCSKeywordSelector?java=false&mode=
keyword
  30:  4.94%: 23/Feb/00 09:19:   /servlet/MCSKeywordSelector?java=false&mode=
category
 167: 19.04%: 23/Feb/00 17:18: /
  80:  6.76%: 23/Feb/00 09:20: /servlet/MCSDisplay
  99: 11.79%: 23/Feb/00 17:15: [not listed: 16 files]
-------------------------------------------------------------------------------

This analysis was produced by analog4.03/Unix.
Running time: 1 second.
```

Appendix D

Online User Questionnaire

I made the following questionnaire available to users of the system through a web form. The questions were numbered, and each possible response is listed with the numerical value which was used to denote that choice when recording the data from the web form. Note that every multiple choice question contains the option "Not applicable". This value was set as the default for every question in the web form so that any bias towards default values would not skew the results.

D.1 MCS Two Minute Questionnaire

Once you have used the this problem-solving archive, we would appreciate it if you would take the time to fill out this brief questionnaire. It should only take two minutes of your time. Mahalo!

If you have not used this archive yet, please check it out and then fill out the questionnaire when you have experienced it.

1. How long have you used the jCVS software?

   - 1. Never
   - 2. Downloaded, installed, or read documentation but never actually used
   - 3. Less than 3 months
   - 4. 3-12 months
   - 5. More than 12 months
   - 0. Not applicable

2. Are you subscribed to the jcvs mailing list?

    - 1. Yes
    - 2. No
    - 0. Not applicable

3. If you are subscribed, on average, how often do you read list messages?

    - 1. Whenever an email arrives
    - 2. Once a day
    - 3. About three times a week
    - 4. Once a week
    - 5. Once a month
    - 6. Almost never
    - 7. I only read it when I have a problem that needs solving
    - 0. Not applicable

4. If you are subscribed, on average, what fraction of list messages do you actually read?

    - 1. Zero
    - 2. Less than a third
    - 3. Between one and two thirds
    - 4. More than two thirds
    - 5. Every message
    - 0. Not applicable

5. Have you used the old archive of the jcvs mailing list?

    - 1. Yes
    - 2. No
    - 0. Not applicable

6. If so, roughly how many times have you used it?

    - 1. More than 10 times

- 2. 6-10 times
- 3. 2-5 times
- 4. Once
- 0. Not applicable

7. How often did you find what you were looking for in the old archive?
    - 1. Never
    - 2. Rarely
    - 3. Sometimes
    - 4. Usually
    - 5. Always
    - 0. Not applicable
8. Roughly how many times have you used this new problem solving archive?
    - 1. More than 10 times
    - 2. 6-10 times
    - 3. 2-5 times
    - 4. Once
    - 5. Never
    - 0. Not applicable
9. How often did you find what you were looking for in this new archive?
    - 1. Never
    - 2. Rarely
    - 3. Sometimes
    - 4. Usually
    - 5. Always
    - 0. Not applicable
10. Since the problem-solving archive has been available, do you find yourself using it instead of the existing archive?

- 1. Yes
- 2. Somewhat
- 3. No
- 0. Not applicable

11. Overall, how would you rate your satisfaction with this new archive?

    - 1. Completely Satisfied
    - 2. Somewhat Satisfied
    - 3. Somewhat Unsatisfied
    - 4. Completely Unsatisfied
    - 5. Undecided
    - 0. Not applicable

12. The messages in this archive were condensed by a human editor, requiring some effort. Would you be willing to help maintain this archive as one of many editors on a volunteer basis? [**Note**: this information is for research purposes so answering "Yes" will not commit you to anything]

    - 1. Yes
    - 2. Not sure
    - 3. No
    - 0. Not applicable

13. Would any other mailing lists you are interested in benefit from having this kind of archive? If so, please list them below:

14. If you have any other comments or suggestions about this archive, please let us know!

Appendix E

Raw Questionnaire Results

In this appendix, I provide all the raw data from the web questionnaire provided to users. See Appendix D for the list of questions and what the answer values correspond to. In Table E.1 I show the answers given to the multiple choice questions, and Table E.2 shows the results of the open-answer questions.

Survey #	Q 1	Q 2	Q 3	Q 4	Q 5	Q 6	Q 7	Q 8	Q 9	Q 10	Q 11	Q 12
1	5	1	6	2	2	0	0	0	0	0	0	0
2	2	2	0	0	0	0	0	0	0	0	0	2
3	5	1	3	2	1	3	3	4	5	1	1	1
4	5	1	0	5	1	1	3	3	5	1	1	1
5	4	1	1	5	2	0	0	4	3	0	2	3
6	4	1	1	3	2	0	0	4	2	0	2	2

Table E.1. Raw response data from questionnaire's multiple choice questions

Survey #	Q 13	Q 14
1		if there were a jcvs-announce list, i would unsub from the jcvs list, join the announce list, and make use of your archive when it was announced. because the noise-to-signal ratio on the jcvs list is so high, i never noticed your archive announcement. i'm looking forward to checking it out now :-)
2	Yes, will discuss it with them.	Your product looks good. As a concept I would be VERY interested in a page giving stats (time devoted to editing, etc...) by those that condense, and some personal feedback from those people as far as how difficult they percieve their task. Keep me on what ever mailing list I recieved your notice and I will watch the development of your program. Thank you.
3	The webmacro mailing list (`www.webmacro.org`)	Maybe you could consider using tools that help you in summarizing and tagging text. There are a lot of scientific projects out there, but I'm not sure if it would be easy to get them on an open source basis.
4	GNUJSP, GJT-DEV	I think the effort is tremendous, and the tool a great improvement over what was available. I would be very excited to see further development and support.
5	PHP Mailing List -> `www.php.net`	Neat concept - could be very useful for larger lists like the php or perl or mysql list. A significant part of the message volume is from people asking common questions. People are usually asking the question because they don't know what keywords to use to search the archives for the solution to their problem. (i.e. If you know how to grok widgets, then you probably know that you need to use the foo() function to do this. However, if you don't know how to grok widgets, then you would have to use 'grok and widget' as your keys for searching the archives. Chances are that whoever asked the question last phrased their question in a different fashion - so the previous exchange on the list regarding this topic is very little help to the new user.) ..uh.. why I am bothering to write this - you obviously know this already - that is why you built the system... :P One note - I found that the interface was pretty clunky. Good Luck! PS Remember that the best defense is a good offence - try using foul language in your thesis defense ;)
6		

Table E.2. Raw response data from questionnaire's open answer questions

Bibliography

[1] Nexial Ascend users mailing list searchable archive. <http://www.nexial.com/cgi-bin/ascendbody>.

[2] MCS: Mailinglist Condensation System. <http://csdl.ics.hawaii.edu/Research/MCS/MCS.html>.

[3] Frederick P. Brooks, Jr. The computer scientist as toolsmith II. *Communications of the ACM*, 39(3):61–68, March 1996.

[4] Larry Wall, Tom Christiansen, and Randal L. Schwartz. *Programming Perl*. O'Reilly & Associates, Inc., second edition, September 1996.

[5] Berkeley Software Design, Inc. <http://www.bsdi.com/>.

[6] Brett Wynkoop. BSD/OS FAQ. <http://www.wynn.com/bsdi/bsdi.faq>, August 1998.

[7] Support Net BSDI list archives. <http://www.support.nl/online/bsd.html>. Archive no longer available.

[8] Excite for web servers. <http://www.excite.com/navigate/>.

[9] Nexial Systems BSDi-Users archive. <http://www.nexial.com/cgi-bin/bsdibody>.

[10] jCVS mailing list. <http://www.gjt.org/servlets/MailingLists/ListInfo.html/jcvs>.

[11] jCVS application. <http://www.ice.com/java/jcvs/index.shtml>.

[12] CVS: Concurrent Versions System. <http://www.sourcegear.com/CVS>.

[13] Everett M. Rogers. *Diffusion of Innovations*, chapter 7. The Free Press, fourth edition, 1995.

[14] Yahoo web portal. <http://www.yahoo.com/>.

[15] ICEMail a Java based email client. <http://www.ice.com/java/icemail/>.

[16] Nexial systems mailing list indexes. <http://www.nexial.com/mailinglists/>.

[17] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, and T. Berners-Lee. Hypertext transfer protocol – HTTP/1.1. RFC 2068, Internet Engineering Task Force, January 1997.

[18] Bill Winett. Tracking your visitors. <http://www.hotwired.com/webmonkey/98/16/index2a.html>, 1998.

[19] Philip M. Johnson. Leap: A "personal information environment" for software engineers. In *Proceedings of the 1999 International Conference on Software Engineering*, Los Angeles, CA., May 1999.

[20] Karl Fogel. *Open Source Development with CVS*. The Coriolis Group, October 1999.

[21] D. Crocker. Standard for the format of ARPA Internet text messages. STD 11, Internet Engineering Task Force, August 1982.

[22] analog web server log file analyzer. <http://www.analog.cx/>.

[23] Google search engine. <http://www.google.com/>.

[24] Robert C. Pedersen. Reviewing Internet mailing lists. *The Serials Librarian*, 30(2):27–33, 1996.

[25] Robert C. Pedersen. A quantitative approach to the description of Internet mailing lists. *The Serials Librarian*, 30(1):39–47, 1996.

[26] Internet FAQ consortium. <http://www.faqs.org/>.

[27] Robin D. Burke, Kristian J. Hammond, Vladimir A. Kulyukin, Steven L. Lytinen, N. Tomuro, and S. Schoenberg. Question answering from frequently asked question files: Experiences with the FAQ Finder system. Technical Report TR-97-05, Department of Computer Science, University of Chicago, June 20 1997. Mon, 23 Jun 1997 21:02:34 GMT.

[28] FAQ Finder web site. <http://faqfinder.ics.uci.edu:8001/>.

[29] Mark S. Ackerman and Thomas W. Malone. Answer Garden: A tool for growing organizational memory. In *OIS90*, Filtering, Querying, and Navigating, pages 31–39. ACM Press, 1990.

[30] Mark S. Ackerman and David W. McDonald. Answer Garden 2: Merging organizational memory with collaborative help. In *Proceedings of the ACM 1996 Conference on Computer Supported Work*, pages 97–105, New York, November16–20 1996. ACM Press.

[31] Faq-O-Matic web site. <http://www.dartmouth.edu/~jonh/ff-serve/cache/1.html>.

[32] Open directory project web site. <http://dmoz.org/>.

[33] Slashdot web site. <http://slashdot.org/>.

[34] Experts exchange web site. <http://www.experts-exchange.com/>.

[35] KnowPost web site. <http://www.knowpost.com/>.

[36] Information markets company web site. <http://www.infomarco.com/>.

[37] Hotdispatch web site. <http://www.hotdispatch.com/>.

[38] Inforocket web site. <http://www.inforocket.com/>.

[39] Exp.com web site. <http://www.exp.com/>.

[40] Expertcentral web site. <http://www.expertcentral.com/>.

[41] Questionexchange web site. <http://www.questionexchange.com/>.

[42] Ithority web site. <http://www.ithority.com/>.

[43] Expertcity web site. <http://www.expertcity.com/>.

[44] Nowonder web site. <http://www.nowonder.com/>.

[45] Terry Winograd. A language/action perspective on the design of cooperative work. *Human-Computer Interaction*, 3(1):3–30, 1987-1988.

[46] T. Winograd. Where the action is (groupware). *Byte Magazine*, 13(13):256A–258, December 1988.

[47] Lucy Suchman. Do categories have politics? The language/action perspective reconsidered. In *Proceedings of the Third European Conference on Computer-Supported Cooperative Work*, pages 1–14, 1993.

[48] C. V. Bullen and J. L. Bennett. Learning from user experience with groupware. In *Proceedings of the Conference on Computer Supported Cooperative Work (CSCW '90)*, pages 291–302, Los Angeles, California, 1990. ACM Press.

[49] R. P. Carasik and C. E. Grantham. A case study of CSCW in a dispersed organization. In *Proceedings of ACM CHI'88 Conference on Human Factors in Computing Systems*, Organizational Issues on Effective Use of Interfaces, pages 61–66, 1988.

[50] JavaServer Pages. <http://java.sun.com/products/jsp/index.html>.

[51] Webmacro java servlet framework. <http://www.webmacro.org/>.

[52] Open Source web site. <http://www.opensource.org/>.