

Scribe

4th Year Final Project Report

Adrian McLaughlin

A project report submitted in part fulfilment of the degree of BSc.
(Hons.) in Computer Science with the supervision of Dr David Wilson
and moderated by Dr Neil Hurley.



Department of Computer Science

University College Dublin

05 March 2003

Table of Contents

1	Abstract.....	4
2	Introduction.....	4
3	Background Research.....	5
3.1	Information Retrieval	5
3.1.1	Indexing.....	5
3.1.2	Searching	5
3.1.3	Stop lists.....	6
3.1.4	Stemming	6
3.1.5	TF x IDF	6
3.1.6	Evaluation.....	6
3.2	User Environment	6
3.2.1	Programming language	6
3.2.2	Mail Client.....	7
3.2.3	XML Document Object Model (DOM).....	7
3.2.4	IR integration	7
4	Approach.....	7
4.1	Mail Client	8
4.2	Assisting the User.....	8
4.3	Displaying the information in Mail threads	9
5	Design	9
5.1	Design of the Email composition-assisting component.....	9
5.1.1	Forming Queries from email contents	10
5.2	Mail Threads	10
6	Implementation.....	12
6.1	Libraries	12
6.2	Mail Client	12

6.2.1	Saving Scribe settings	12
6.2.2	The Directory Structure of Scribe.....	13
6.2.3	The email table	14
6.2.4	Composing a new message	14
6.2.5	Downloading emails	14
6.2.6	Replying to a received email	14
6.2.7	Communication with the User.....	14
6.3	Information Retrieval Component.....	15
6.3.1	Email composition-assisting component.....	15
6.3.2	Mail threads	17
6.3.3	Class Diagram of Scribe	18
7	Testing and Evaluation.....	19
7.1	Using a restricted email set	19
7.2	Validation using a large set of emails	21
7.3	Group or User testing.....	21
7.4	Known Issues	21
8	Future Work	21
9	Conclusion.....	21
10	Acknowledgements.....	22
11	Bibliography	22

1 Abstract

This project is to see in what ways an email client can assist the user with their vast email collection and make the information contained within more available to the user. The issues of how the information is conveyed to the user and if the information is of any use to the user are main areas of interest in this project. The report contains the background sources on useful techniques and how they were applied to create a mail client.

2 Introduction

With increasing communication over the Internet, a person will quickly find themselves overloaded by information. This information overload is especially true in the case of email, in communication with mailing lists, friends and work for example a large collection of emails can be obtained. Important information can so easily be obscured by the overwhelming quantity of irrelevant emails.

This project is concerned with helping to manage this collection and making the information contained within it more available and useful to the user. This involves providing a mail client that incorporates scheduling, machine learning, information extraction, as well as natural language and discourse processing techniques. One area of interest is in the creation of an email-composing assistant. The aim of the assistant is to see if old emails can assist the user in the composition of new emails. Old emails contain old conversations, topics, and information about people, places and experiences that can inspire or remind the user of items of relevance when composing a new email. For example, when writing to a friend about topics you already written to your other friends about, it would be useful to have these emails to hand to help shape the new email, so there's less chance of forgetting something important or mentioning things from one group of people to another that don't share the same interests.

In other consideration of design, the assistant must assist without disrupting or distracting the user from their composition task. Another area of interest is the organisation of emails into groups of the same context and into message threads i.e. the change of messages in a conversation style

The area of basic Information Retrieval has been researched as it's the main technique that going to be employed. Research papers pertaining to other peoples attempts at applying IR with user interactive interfaces have also been studied, to see what they had learned and if this is applicable to this project.

Section 3 covers the research undertaken in the development of Scribe, Section 4 and 5 covers the approach and design of the email-composing assistant and the techniques to be employed for email organisation. Section 6 covers the detailed design of Scribe's components that is the mail client, email-composing assistant and email organisation and how they are integrated together. Section 7 outlines the testing and evaluation of the overall system.

3 Background Research

3.1 Information Retrieval

To retrieve relevant information from emails collected, it requires the techniques of Information Retrieval. To find information that is relevant to a user, the user must submit a query. A query being a word or a string of words that characterises the information that the user seeks [1]. To make the task of searching the emails easier and quicker, the emails are pre-processed before a query is even made.

3.1.1 Indexing

The emails are indexed; there are many ways to do this. Two ways of doing this, is using inverted files and signature files. An inverted file is a representation for a group of documents (in this case emails) that is essentially an index. For each word that appears in the group of documents, an inverted file lists each document where it appears. A signature file is a representation of a group of documents where the documents are hashed to a bit string. This is essentially a compression technique to permit faster searching. [1]

3.1.2 Searching

To find information relevant to the query, emails similar to the query must be found. There are a few ways to achieve this; two ways are the Vector space model and Probabilistic model. In the Vector Space Model, the group of emails and the query are mapped as vectors in an n th dimensional space. The closest document to the query is the most similar and the similarity between the two is the measure of cosine of their vectors known as the cosine similarity metric. There are other similarity metrics like the Simple, Dice, Jaccard, and Overlap but generally none is better than the other over all types of queries [5]. In the Probabilistic Model retrieved documents are ranked according to a probability of relevance.

Similarity Measures:

Query =Q

Document=D

$|Q \cap D|$ Simple Matching returns a score of one for documents that exactly match the query.

$$2 \frac{|Q \cap D|}{|Q| + |D|} \quad \text{Dice}$$

$$\frac{|Q \cap D|}{|Q|^{\frac{1}{2}} * |D|^{\frac{1}{2}}} \quad \text{Cosine}$$

$$\frac{|Q \cap D|}{|Q \cup D|} \quad \text{Jaccard}$$

$$\frac{|Q \cap D|}{\min(|Q|, |D|)} \quad \text{Overlap}$$

3.1.3 Stop lists

Not all words will help in finding a similar email. Words like 'an', 'in', 'of' and 'the' are useless in a query as they appear frequently in most documents. Usually these words are added to a list called a stop list and are removed when indexing a document or parsing a query.

3.1.4 Stemming

Another technique used to reducing the index size and aid better searching is stemming. Words like 'effective' and 'effects' has a common base word 'effect' which can represent both terms in the index hence given a generally small index which won't take long to search. A problem can occur when a word is incorrectly stemmed like 'army' to 'arm' which has a completely different meaning. The most well know stemmer is the 'Porter Stemmer'. It uses a set of heuristics to stem words. For example plural words that end with 'ies' has a stem which ends with 'y' as in 'fairies' and 'fairy' [2].

3.1.5 TF x IDF

Other words may be also useless or less important in finding a relevant document but may not be obvious as words on the stoplist. If for example, the documents are computer science reports then including words like 'computer', 'maths' or 'user' would be pointless as the majority of the reports would contain these terms in high frequencies. A weighting scheme can be employed to help target more relevant documents. One weighting scheme is called TF x IDF. Term Frequency (TF) is the number of times a word appears in a document. Inverse domain frequency (IDF) provides high values for rare words and a low value for common words. With this weighting scheme a word that is frequently appears in one document but rarely appears in other documents is given a higher value than a word that is frequent in all or most documents.

3.1.6 Evaluation

An important factor to consider is evaluating the results of any retrieval or search. There are two main important measures, Precision and Recall. Precision is the proportion of retrieved documents actually relevant. Recall is the proportion of relevant material actually retrieved.[3] Precision is desired as it gives the closest to the best answer to the query but Recall is desired as it gives the user more information and information that is a tangent to the query offering the user scope to search.

3.2 User Environment

3.2.1 Programming language

To implement the strategies of chapter 4, an email client had to be implemented, Python and Java were both considered, Python is faster, easier to use than Java and good at parsing text files but Java had a fully implemented IR and Mail library. So Java was chosen, as it would save time on the creation of a mail client.

3.2.2 Mail Client

The project requires a mail client that can send and receive email and can store email locally. Python's email classes and JavaMail [16] were researched to find the most useful toolkit with which to build the mail client.

JavaMail is an API, which allows retrieving and sending to remote mail servers. To facilitate the local storing of retrieved email a library from icemh.org [17] was employed which extends the function of certain JavaMail classes.

3.2.3 XML Document Object Model (DOM)

The DOM is a tree structure, which can embody an XML file. Each node of the tree corresponds to one of the components from an XML Structure. DOM functions allow you to manipulate this tree structure.

Many programming languages provide libraries to create a DOM from a XML file, parse it and write the DOM to a XML file.

3.2.4 IR integration

The integration of the mail client and the IR system and the user interaction with the system as a whole had to be considered. The paper on the 'Remembrance Agent' by Rhodes and Starner gives account of their attempt to integrate an IR system with a GUI. Basically their program displayed a list of documents that could be relevant to the user's current context of the piece of text they were writing. Some of the key concepts that they portrayed were that the agent should run continuous and the suggestions made to the user are unobtrusive and not distracting. The remembrance agent consists of two parts. A front end that continuously monitor certain user action like what the user is typing and conveys this information to the back end which retrieves relevant documents and passes them to the front end to display to the user. The way it displays this information is as one-line summaries at the bottom of the window. Another interesting technique it used to keep the retrieved documents relevant to the user was how it forms a query from the user's text. As an example given by their paper, the first retrieved document was based on the last 500 words in the user's document, the next two on the next 50 lines and the last on the last ten lines. This keeps the retrieved document relevant to the users overall context and on any digression that the user may of taken. In the retrieval of emails, we hope to employ a similar strategy for query forming from a body of text, in our case the contents section of an email.

The paper 'Watson' is similar to the 'Remembrance Agent' but different in that it proposes an Information Management Assistant (IMA) that learns from the user's interaction with the retrieved information to hopefully learn user behaviour and retrieves better results in the future.

4 Approach

The main design issues of Scribe are its email-composition assistant, displaying of mail threads and providing of an email environment. The functionality that needed to be designed into email-composition assistant is the ability to form queries from the contents of the email being composed, retrieve emails similar in context to the email being composed and display these emails to the user. In the case of mail threads, it needs the ability to display the threads of conversation by the way of replies running through the emails and group the emails from the

same mailing-lists/ newsgroups. The next sections covers our approach to achieving these functionalities in Scribe.

4.1 Mail Client

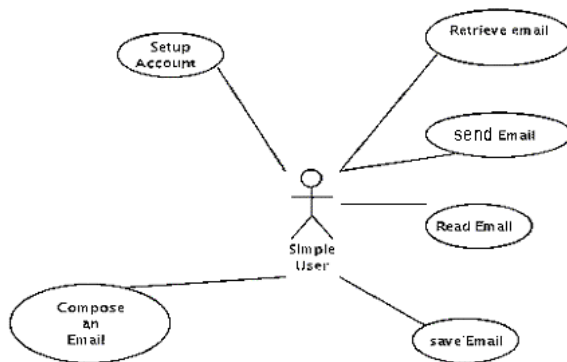


Figure 1. Use case diagram for the mail client

4.2 Assisting the User

The emails need to be stored and retrieved at runtime to be displayed to the user. The technique that was chosen was Information Retrieval as it's efficient in storage and retrieval but also the collection of emails can be efficiently searched. As the user is composing a new email, the system would retrieve and display to the user, similar emails from the emails already stored. It's hoped these emails would assist the user in their task of composition. The system would have to achieve this without disrupting the users activity and without distracting the user. The emails retrieved would have to be close to the context of the section of the email the user is composing. The strategy that was employed, was for a thread to grab the text from the email that the user was composing and create a query that could be used to retrieve similar emails. The results, if any can be displayed in a separate window, where the user can view the emails and their contents. The thread allows the user and system to work concurrently so not disrupting the user and the separate window was used so not to distract the user. To keep the retrieved emails similar to the context of the new email, the query formed from this email must be manipulated to favour its context; one strategy employed was to take the last ten words of the email to form a query. Another strategy was to divide the email in quarters and weight the sections in increasing weights in the query. The idea behind the second strategy is that first strategy would only retrieve emails similar to the sentence or paragraph the user is currently composing, but what if there are no emails similar in context, the second strategy addresses this shortfall by retrieving emails similar in context to the overall context of the new email though still valuing higher, sections near the end of the new email.

4.3 Displaying the information in Mail threads

Mail threads in emails were to be displayed to the user; the technique employed here was to use the references and reply-to headers in the emails to trace the threads. When a reply is created to an email the original emails message I.D. number is inserted into the reference header. When users are mailing to a mailing list, the reply-to header usually contains the mailing-list address.

5 Design

5.1 Design of the Email composition-assisting component.

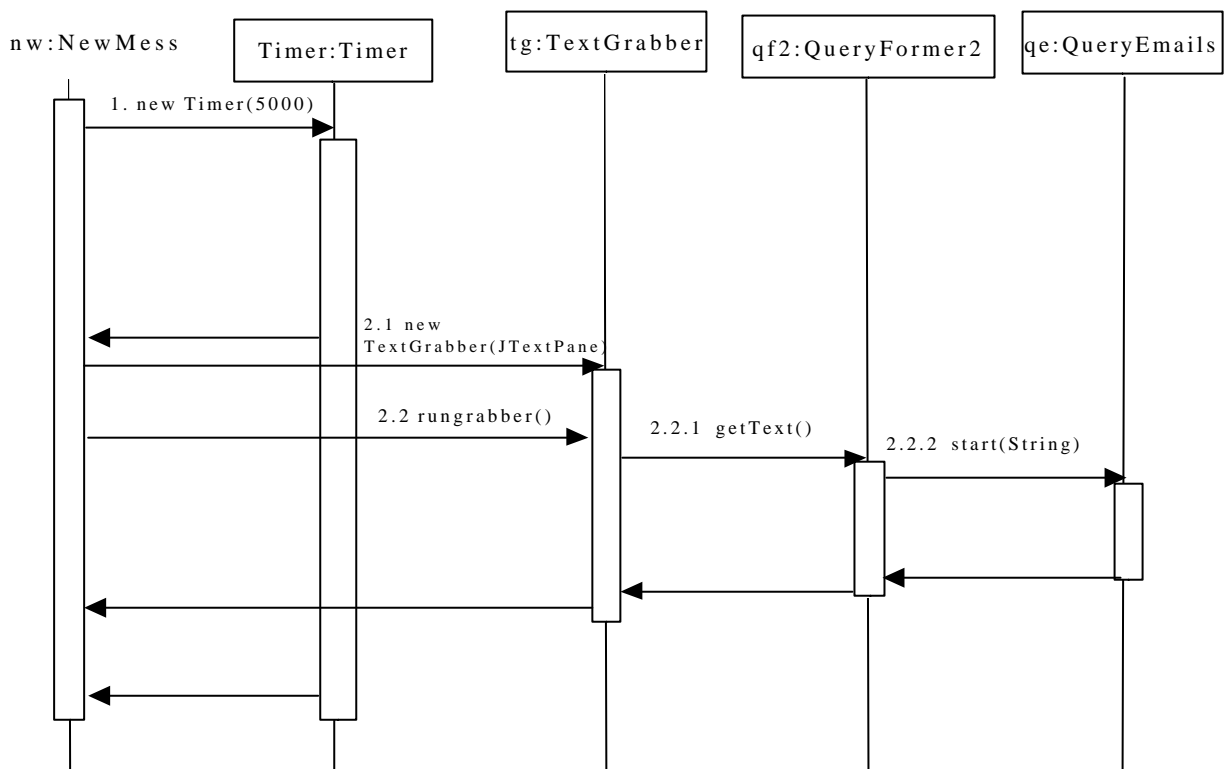


Figure 2. Sequence diagram of the Email composition-assisting process.

The email is composed in an editor panel which is embedded in a window as in the case of Fig 1. the window is called 'NewMess' . When this window is created, a timer is started which starts a separate thread of execution 'TextGrabber'. The editor panel is shared between the thread and window so to reduce user interference. The thread takes the text in the editor panel passes it to another class 'QueryFormer2' which converts this text into a weighted query. This query is then passed to 'QueryEmail ' class, which searches against the stored emails for similar emails. The

results are a list of emails, which are passed back through the previous classes and displayed in the original window in a separate panel.

5.1.1 Forming Queries from email contents

The first strategy employed was to take the last ten words in the email to form the query.

Email extract:

“In any case, the democratic party still has the same problem, no real worthy candidates and no real message The only two I know of are Lieberman and Kerry and I really see either of them having a snowball's chance in hells, far as the message, I'm pretty much a Democrat and I have no idea hat the message of the Democratic party is these days”

Example of Query forming strategy no.1:

The last ten words are taken from the text as a basis for a new query. Their order are reserved but this is a side effect from the extraction and has no effect on the Information Retrieval process. The words undergo Porter Stemming to remove plurals and reduce words to their stems. Stops words are also removed.

Query formed from email:

(dai parti democrat messag idea and democrat a pretti messag)

Note: The query has under gone Porter stemming and the removal of stop words.

The first strategy when first tested didn't retrieve many context similar emails, so a second strategy was formulated.

Example of Query forming strategy no.2:

Query formed from email:

The email contents was taken and divided into four and a higher weighting was given to the section near the end so to retrieve emails with similar context to the section of the email to which the user is composing but also to retrieve emails with context similar to the overall email.

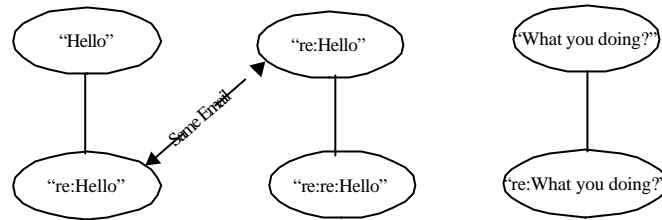
(dai^4.0 parti^4.0 democrat^4.0 messag^4.0 hat^4.0 idea^4.0 and^2.0 democrat^2.0 a^2.0 pretti^2.0 messag^2.0 far^2.0 hell chanc snowbal a really)

Note: The query has under gone Porter stemming and the removal of stop words and the symbol '^' implies the weighting of each term.

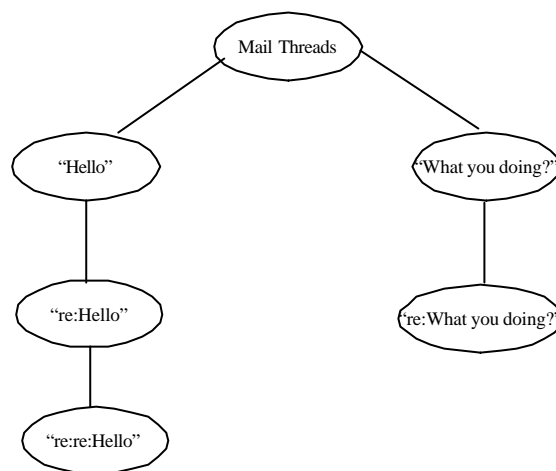
5.2 Mail Threads

The first attempt to display mail threads involved searching the stored emails for emails with 're:' in the beginning of their subject line as this denotes them as replies. To find the original emails to

which these are replies to, another search was conducted with the replies subject line minus the 're:' segment. These were stored as tree structures. If the same email existed in different trees, these trees were merged together. In the end it would result in a single tree structure.



Email replies and their original emails



The tree generated after combining all the sub-trees

Figure 3. Simple generation of mail threads

This was too simplistic and had inherent problems. Like not all replies have 're:' in their subject lines, emails may have the same subject lines but may not be the same email. After studying a large collection of emails from newsgroups it was obvious that the headers in the emails stored the nature of the email, especially the headers 'references' and 'reply-to'. When a reply is created to an email the message id number of the original email is stored in the reference header of the reply. The message id number is a serial that is stamped on an email on its creation it supposed to be unique. Using this knowledge, a class was created to search emails containing the reference header. Using the message id number stored in the header to find the original email if it has been stored. Then generating a tree structure similar to my first attempt.

In testing it was noted, that emails from newsgroups or mailing lists have the mailing list or newsgroup address inserted into the reply-to header so that a reply to an email goes to the mailing list or newsgroup and not to the author of the email. The information was used to group emails together from the same newsgroup or mailing list.

6 Implementation

6.1 Libraries

The libraries used for Scribe were JavaMail [16], Lucene [9], IceMH [17]. JavaMail library provide a toolkit for creating emails and receiving and sending to mail servers. IceMH extended the functionality of JavaMail by allow emails to be stored locally. Lucene library was used to implement an Information retrieval system.

6.2 Mail Client

The functionality that was implemented in the mail client:

- The sending of emails
- The receiving of emails
- The reading of emails
- The configuring the server settings

6.2.1 Saving Scribe settings



Figure 4. Configuration window

The values of mail host, username, password and email address are need to stored, as they are different to each user but generally don't change. The values are stored locally in an XML file that can be read at the start-up of the program. The 'userpref' class creates a DOM tree [19] from the XML file. The 'account' class holds the values of mail host, username, and password and the 'user' class is comprises of an account and the values of email address. Taking for an example a user called Bob with an email account on server.ucd.ie. He would enter his username and

password to allow authentication to server.ucd.ie which would be entered in the next section and finally his email address so Scribe can place in any new email the person that created it.

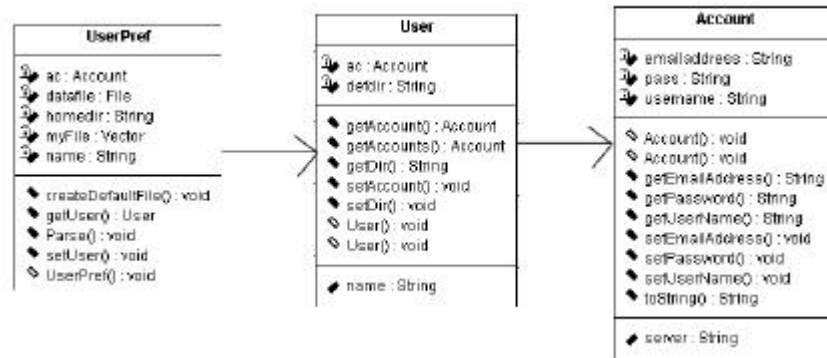


Figure 5. Class Diagrams for the definition of a user.

When values are changed in the configuration window fig. 6 pressing the 'ok' button will write them to the XML file and update the dependant classes.

6.2.2 The Directory Structure of Scribe

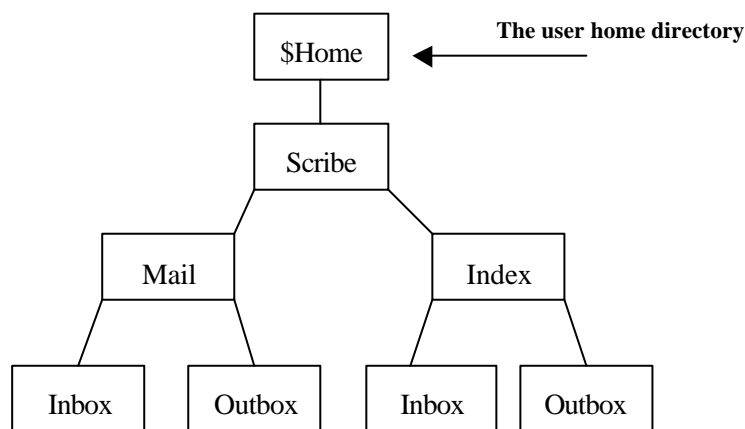


Figure 6. Directory Structure of Scribe

The directory structure of Scribe is used to allow the quick deployment of Scribe to different users and computers. Also with a few modifications allow the deployment to a Unix environment. Before this was implemented, all directory locations were hard-coded in the program, so when the program was moved to a new computer environment, these lines of code had to be updated requiring time and causing run-time errors. In the case of our hypothetical user Bob, if Bob's home directory was '/home/bob' then Scribe would know it should be in the path or folder '/home/bob/Scribe/' and the emails can be found in '/home/bob/Scribe/Mail/' and the Lucene index in '/home/bob/Scribe/Mail/Index'.

6.2.3 The email table

The email table as in the Inbox window of Scribe displays certain information from each email to the user. It hides from the user all the other headers of the email that most users won't be concerned with. The table displays, the sender, the recipient, date the email was sent, the format of the email, the subject and the status of the 'read' flag of the email.

6.2.4 Composing a new message

To open up the composition window, the 'new' menu entry has to be selected in the file menu. The values for recipient, subject and contents of the new email can be edited here. The clicking of the 'send' button saves the message to the outbox folder and transports the message to the server specified in the XML file using the Simple Mail Transport Protocol. In the case of Bob, the new email would be transported to server.ucd.ie, which in turn forward it on to the correct address and a copy would be saved in '/home/Scribe/mail/Outbox'.

6.2.5 Downloading emails

Pressing the 'pop' button sends a request to the mail server specified in the configuration window for new emails. The POP3 protocol is used because it's simple and in common use. If there are any new emails, they are downloaded, saved to disk and the inbox window is updated. When the transaction is over, the connection to the mail server is terminated.

6.2.6 Replying to a received email

Right clicking on the desired email brings up a popup menu, on which is a reply entry. Selecting this opens up the composition window. The values from the 'from' header and subject line are inserted into the recipient and subject boxes respectively. What is not is that the message id of the original email is inserted into the 'reference' header of the reply.

6.2.7 Communication with the User

When a problem occurs a window is displayed to the user tell about any problems that have arisen. Since most of the problems the window relays are important, it displays over the client itself, to grab the attention of the user.

Some problems it would relay is:

- If the mail server specified doesn't exist.
- If the program couldn't get authorisation to connect to a server.
- If there is a problem with an email.
- Any problem with the program that could be used for debugging

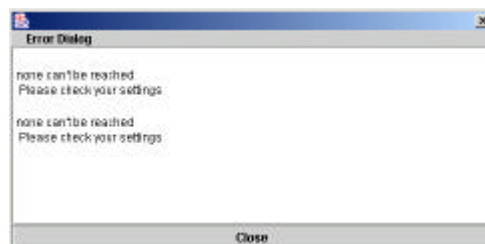


Figure 7. Error Dialog reporting a problem with the settings given by the user.

6.3 Information Retrieval Component

A basic information retrieval system has implemented using the Lucene API [9]. When the emails are saved to the local mailbox they are also indexed and this index is stored in a separate directory. When this index is to be searched a query must be created to which Lucene returns an array of emails satisfying the query in the order of their similarity.

6.3.1 Email composition-assisting component

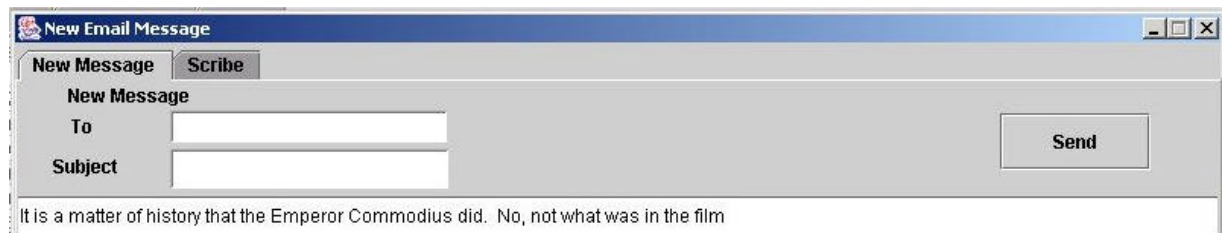


Figure 8. Email being composed

Subject	Sentby	Date	rank
Re gladiator, my oh my	chil-out@ix.netcom.com	16 Dec 2002 16:08:49 GMT	
Re Film that makes you emotion...	lgalbrait@ozonline.com.au	17 Dec 2002 02:00:28 GMT	
YES DIANE LANE WINS NY FIL...	Allegro_82@webtv.net	17 Dec 2002 04:19:58 GMT	
NY Film Critics upset Diane Lan...	kasmith@uselesspamaccess1....	17 Dec 2002 04:27:07 GMT	
Re Fave books you think should ...	jgh@attcanada.ca	17 Dec 2002 02:48:35 GMT	
Re YES DIANE LANE WINS NY...	jgh@attcanada.ca	17 Dec 2002 05:15:30 GMT	
Re Film that makes you emotion...	-fake-@email.com	17 Dec 2002 06:28:22 GMT	
Gangs of New York on History Ch...	kilroybass@catlover.com	17 Dec 2002 05:52:34 GMT	
Re let us liberate Iraq and not be...	chil-out@ix.netcom.com	17 Dec 2002 02:12:38 GMT	
Re American Foreign Policy Aga...	chil-out@ix.netcom.com	17 Dec 2002 02:01:05 GMT	

Figure 9. Search results

A JProgressBar is used in the rank column to easily show the relevance or similarity of each email to the query.

Figure 1 describes the sequences of events of the Email composition-assistant.

TextGrabber class is the thread that is generated. It shares the common data structure that holds the emails contents.

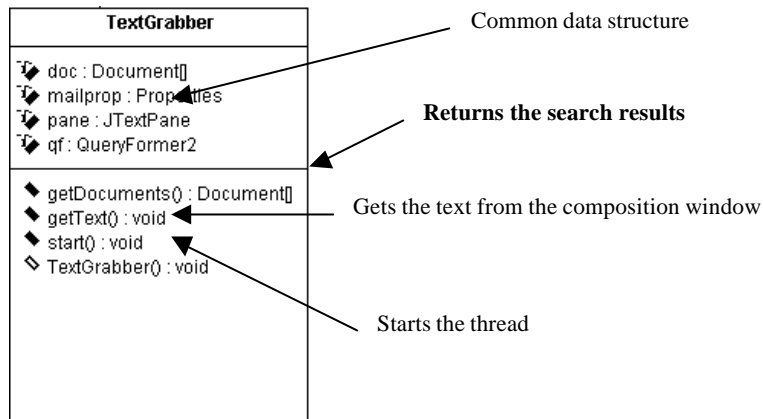


Figure 10. Class diagram of text grabber

QueryFormer class employs the second strategy discussed in section 5.1.1 to generate a query from the contents of an email.

QueryEmails class performs the search on the email index and returns the search results. It embodies the IR aspect of the program and provides a simple interface to the rest of the program.

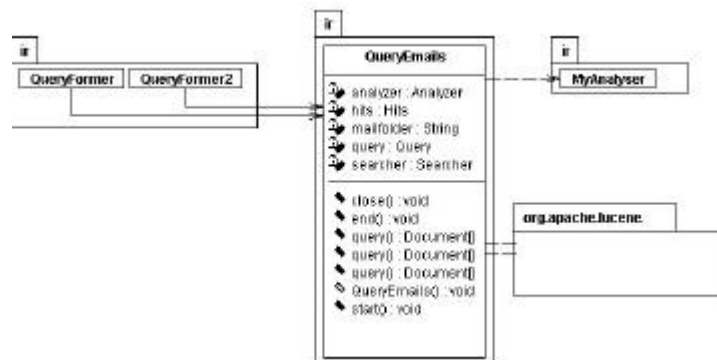


Figure 11. Class Diagram of QueryEmails

6.3.2 Mail threads

Threads generated from reference header in emails

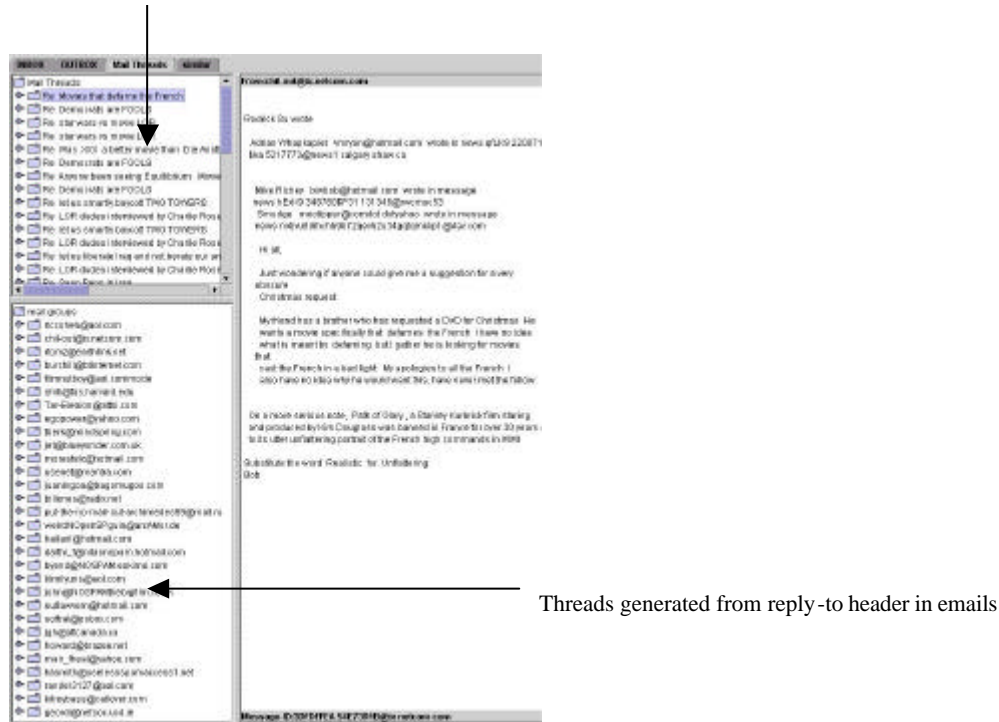


Figure 12. Mail Threads panel in Scribe

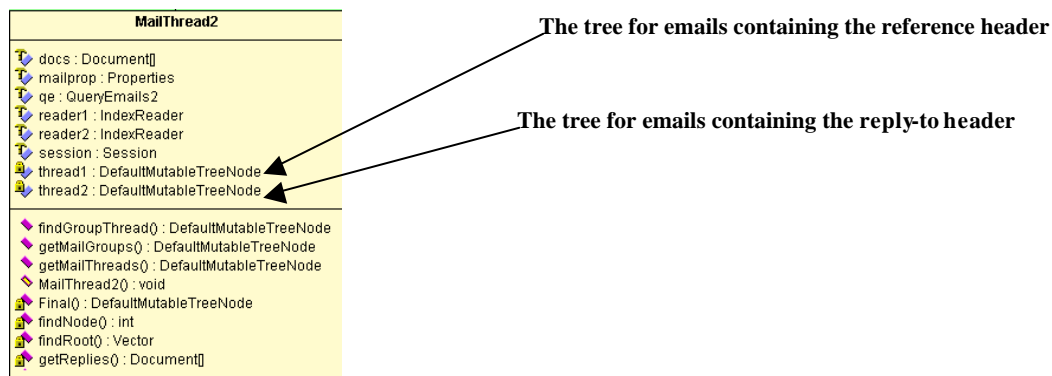


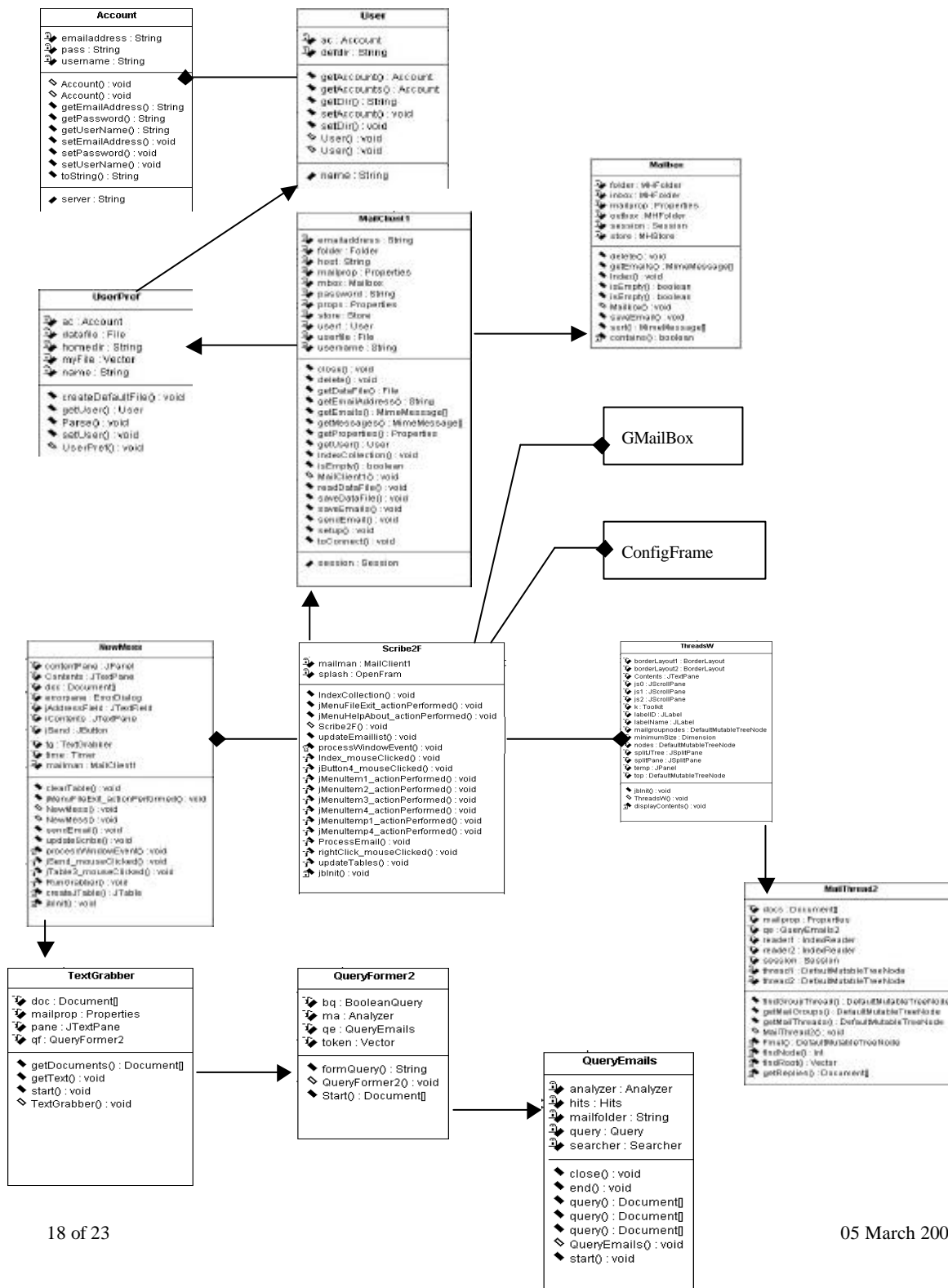
Figure 13. Class Diagram of Mail threads

Using the email Index, the class finds all emails with the reference header. Using this set of emails, the original emails to these replies are searched for by using the message id numbers stored in the reference section of these replies. A tree is constructed and shown as a 'JTree' in the top-left panel of the Mail threads window. Using the values stored in the reply-to header a second tree is constructed to show newsgroups and mailing lists and shown as a 'JTree' in the bottom-left panel of the Mail threads window. A tree structure is the best way to represent mail threads, as an

email can have replies and those replies can in turn have replies, this structure can be superimposed on a tree for clarity.

A wrapper class ‘VisualNode’ was create to select the values from the emails to be show in the ‘JTree’s.

6.3.3 Class Diagram of Scribe



7 Testing and Evaluation

It is necessary to validate claims made and test that the functionality of Scribe components, that they meet the specification in the design. A restricted email set was used to test if the email-composition assistant was retrieving emails similar in context to the email being composed. Using a small-defined number of emails restricts possible contexts unlike a newsgroup based on topic but the emails usually digress. A large email set was used to verify that Scribe can handle emails without problems as email programs produced by different companies rarely construct emails conforming to any single standard. This was also used to test if Scribe could display the mail threads in the email set. Group testing was used to test if Scribe's features are useful to a user.

7.1 Using a restricted email set

A set of ten emails that were based on different topics were created and placed in the index of Scribe. An email was composed that had sections or sentences focused on different topics covered by emails in the indexed. Each section of the email was pasted into the composition window of Scribe and the results returned by Scribe were recorded for each section. The results of this test showed the query formed from the email contents wasn't retrieving context similar emails. The strategy of taking the last ten words for the query didn't work well. So this led to the creation of another strategy, which involved weighting the terms in the whole of the contents of the email and using this as the query. The same experiment was repeated but using this strategy to form the queries. The results of the experiments showed the second strategy was better than the first at retrieving context similar emails.

The email used to generate the search results:

"Hello Steve,

I think of going on a holiday to the Brecon Beacons, I have an aunt who lives there. We haven't seen in a while after her accident with a frying pan, her legs were terribly burned. I really going for the sky diving. I still have to do that essay for History class on late 20 century Russian politics, How positivately boring! I got a new computer it came with that awful operating system you know the one, I formatted it and set up a Unix system, I feel so much better after doing that like an exorcist who has driven off the demons. My brother is working in Greece as a consultant for the government dept of the marine. He says that it's good to get away from the rain."

Mike.

Query Syntax	Actual Query	Intended Document	First Three Ret. Docs.	Rank of Intended Doc	No Retrieved
+(term1^4 term2^2 term3 term4 subject:(last five terms)^5	+(contents:becon^4.0 E2contents:breacon^4.0 contents:holidai^2.0 contents:a^2.0 contents:steve contents:hello subject:becon^5.0 subject:breacon^5.0)	1009	1009,1001	1	2
=(term1^4 term2^2 term3 term4 subject:(last five terms)^5	+(burn^4.0 terribl^4.0 leg^4.0 pan^4.0 fry^4.0 a^2.0 accid^2.0 a^2.0 live^2.0 aunt^2.0 beacon brecon holidai a steve hello subject:burn^5.0 subject:terribl^5.0 subject:leg^5.0 subject:pan^5.0 subject:fry^5.0)	1007	1007,1010, 1009	1	5
+(term1^4 term2^2 term3 term4 subject:(last five terms)^5	+(dive^4.0 sky^4.0 realli^4.0 burn^4.0 terribl^4.0 leg^4.0 pan^2.0 fry^2.0 a^2.0 accid^2.0 a^2.0 live^2.0 aunt beacon brecon holidai a steve hello subject:dive^5.0 subject:sky^5.0 subject:realli^5.0 subject:burn^5.0 subject:terribl^5.0 subject:leg^5.0)	1008	1008,1007,1010	1	6
+(term1^4 term2^2 term3 term4 subject:(last five terms)^5	+(bore^4.0 positivit^4.0 polit^4.0 russian^4.0 centuri^2.0 20^2.0 late^2.0 class^2.0 histori^2.0 essai dive sky realli burn subject:bore^5.0 subject:positivit^5.0 subject:polit^5.0 subject:russian^5.0)	1003	1003,1007,1005	1	6
+(term1^4 term2^2 term3 term4 subject:(last five terms)^5	+(demon^4.0 driven^4.0 exorist^4.0 better^4.0 feel^4.0 system^4.0 unix^4.0 a^4.0 set^4.0 and^4.0 format^4.0)	1001	1001,1002,1007	1	6

Figure 14. Experiment using the first query forming strategy

Query Format	Actual Query	Intended Document	First Three Ret. Docs.	Rank of Intended Doc	No Retrieved
+(a b c)	+(becon brecon holidai go think i steve hello)	1009	1001, 1004, 1009	3	3
+(a b c)	+(burn terribl leg pan fry accid seen we)	1007	1007, 1009, 1002, 1006, 1001	1	5
+(a b c)	+(dive sky go realli i burn terribl leg pan frv)	1008	1007, 1004, 1001, 1009, 1002	0	5
+(a b c)	+(bore positivit how polit russian centuri 20 late class histori)	1003	1001, 1007, 1002	0	3
+(a b c)	+(demon driven exorist better feel i system unix)	1001	1001, 1002, 1009, 1008	1	4
+(a b c)	+(rain awai get good sai he marin dept govern)	1002	1001, 1009, 1002	3	3

Figure 15. Experiment using the second query forming strategy

7.2 Validation using a large set of emails

Emails were extracted from a newsgroup 'alt-movies'. About five hundred and fifty. These were used to test Scribe's ability to handle different styles of emails and display the mail threads. The lessons learnt here is never count on anything being in an email, for example emails having no sender address, emails with no message id numbers.

7.3 Group or User testing

A group of four people were selected to test the program, with the aim of testing the message composition and mail threads components of the program. This aim couldn't be achieved due to time constraints and the number of new bugs found during this experiment. One bug caused the program to crash when the email didn't have message id number. Another bug stop the email download from server process when email with unknown encoding was encountered. These problems were all simple to resolve but wasted value time. Though it did show the validation using a large set of emails didn't fully test Scribe's ability to handle different styles of emails.

7.4 Known Issues

Bugs still remain in the program. Two major problems still exist, a mutual exclusion problem with Lucene Index and writing back changed flags to emails, so the changes are stored. These are all solvable problems with time.

8 Future Work

An experiment where a large group of people can validate the claim that the email-composition assistant does help with the composition of email and that the mail thread system is helpful needs to be undertaken.

Since email comes from different groups i.e. friends, work and mail lists you may want to send them the same topics of information. An idea was envisioned based on an address book where you can save texts for different groups, so when you compose a new email for someone else in that group, the texts for that group are displayed to you.

9 Conclusion

In the extent of providing a mail client program, this aim was achieved. Though the program has a few remains bugs but the functionality of simple email program has been achieved. The testing of the email-composition assistant wasn't conducted as time run out for the project. The restricted email set experiment proved the strategy for weighting terms near the end of an email was better for retrieving emails with similar context than the strategy of taken the last ten terms of an email. It also shows the email-composition assistant can provide emails with the correct context to what the user has typed. The graphical displaying of mail threads, newsgroups and mail groups help to

organise emails by grouping relevant emails together. What still needs to be proved is if the email-composition assistant and the mail thread display can still be effective when the email base is scaled up from ten to thousand. We feel Scribe is a step in the right direction on solving the Information overload experience by people on the Internet with the techniques it employs but Scribe is only the beginning of what can be achieved.

10 Acknowledgements

I would like to thank David Wilson for his advice and guidance, the Testing group; Paul McDonnell, Teresa Monahan, Donal O’Kane, Ciaran McNamara for their patience and time. I also like to thank Donal O’Kane and Ciaran McNamara for their helpful advice.

11 Bibliography

1. <http://www.cs.jhu.edu/~weiss/glossary.html> (10/10/2002) Glossary for Information Retrieval
2. www.tartarus.org/~martin/PorterStemmer (10/10/2002) Porter Stemming Algorithm Information website
3. Ray Larson and Marti Hearst, (10/10/2002) www.sims.berkeley.edu/courses/is202/f98/lecture17/slidoor.html Lecture 17 of SIMS 202: Information Organization and Retrieval course. University of California, Berkeley, School of Information Management and Systems.
4. Justin Zobel, and Alistar Moffat (1998) “Inverted files vs. Signature files for Text indexing” A preliminary release of an article accepted by ACM Transactions on Database systems
5. Justin Zobel, and Alistar Moffat (1998) “Exploring the similarity space”, SIGIR Forum
6. William B. Frakes and William Bruce, "Information retrieval: data structures and algorithms ", Prentice Hall, 1992
7. Salton and McGill, (1983)"Introduction to Modern Information Retrieval", McGraw-Hill, New York.
8. Marti Hearst (1998) "Current Topics information Access in Background” Lecture course SIMS 202: Information Organization and Retrieval course. University of California, Berkeley, School of Information Management and Systems.
9. Lucene (24/11/02),Open Source Java search toolkit, Jakarta Project, <http://jakarata.apache.org/lucene>
10. Xapian Project,(24/11/02)Open Source Probabilistic Retrieval library <http://xapian.org/>

11. J.Bradley, Thad Starner (1996) "Remembrance agent", in Proceedings of the First International Conference on the Practical Application of Intelligent and Multi Agent Technology.
12. J. Budzik, Kristian Hammond (1999) "Watson: Anticipating and Conceptualizing Information Needs" in Proceedings of the Sixty-second Annual Meeting of the American Society for Information Science.
13. J. Budzik and K Hammond (1999), "Q&A: A system for the Capture, Organization and Reuse of Expertise", In Proceedings of the ASIS 1999 Annual Conference.
14. Jay Budzik, Kristian Hammond (2000), "User interactions with Everyday Applications as Context for Just in time Information Access", Proceedings of the 2000 International Conference on Intelligent User Interfaces
15. David B. Leake, Ryan Scherle, Jay Budzik, Kristian Hammond (1999) "Selecting Task-Relevant Sources for Just-in-Time Retrieval" In Proceedings of the AAAI-99 Workshop on Intelligent Information Systems, Menlo Park, CA
16. www.javamail.org (10/2002) JavaMail Mail Client toolkit implemented in java
17. www.icemh.org (02/2002) local mailbox toolkit implemented in java
18. Monica Pawlan (2001) "Multithread Swing Applications", Java tutorials at <http://java.sun.com>
19. Eric Armstrong (21 Aug 2001) "Working with XML: The Java API for XML Parsing (JAXP) Tutorial"