

Ishmail: Immediate Identification of Important Information

Jonathan Isaac Helfman
AT&T Bell Laboratories
600 Mountain Avenue
Murray Hill NJ 07974
jon@research.att.com

Charles Lee Isbell
MIT Artificial Intelligence Laboratory
545 Tech Square
Cambridge MA 02139
isbell@ai.mit.edu

ABSTRACT

This paper describes Ishmail, a program designed for people who get a lot of electronic mail. Most email programs do not address the main problem experienced by people who get a lot of email: information overload. Given a deluge of email, how does one maintain control over incoming message traffic and reduce the time required to find important messages? Some email programs support classification of messages into separate mailboxes, but this is only a partial solution. Ishmail is unique in that it not only sorts messages into mailboxes, but it orders mailboxes by a combination of user-specified priorities and alarms. While most mail programs only alert users about unread messages, Ishmail supports independent alarms on each mailbox with customizable thresholds and filters. Users control their alarms, mailboxes, and messages through customizable summaries that act as both views and interactive controls. Three additional unique features of Ishmail are 1) the ability to read messages safely from multiple locations, 2) multiple levels of archiving, and 3) multiple levels of customization (Ishmail exploits Emacs Lisp's extensibility and customization features). Ishmail has been in continuous use by about a dozen people for more than a year and by several dozen people for several months. Many users have commented that now that they have Ishmail, they cannot imagine living without it. This paper describes the design and implementation decisions that we think make Ishmail useful, in particular, Ishmail's user-interface strategy and Ishmail's database and classification system. Generalizations about broader information filtering problems are also discussed.

KEYWORDS: alarms, email, filters, information overload, object-oriented database, message archiving, messaging

Introduction

Electronic mail technologies have evolved so that it is now easy to broadcast information to a lot of people. Unfortunately equivalent technologies do not exist to help those people deal with this sudden influx of information. As peo-

ple start to get more and more email, the term information overload becomes less of a buzzword and more of a real problem. Important messages become harder to find as mailboxes overflow. Most email programs make the implicit assumption that the number of messages is small and manageable [5, 16]. Structural information about groups of similar messages or positional information about the location of important messages must be maintained by the user.

One strategy for managing overload is to organize messages into different mailboxes; however, this does not solve the problem. Even email programs that make it easy to organize messages into different mailboxes make the mistake of assuming that the number of *mailboxes* is small and manageable [10, 14]. Information about groups of similar mailboxes or their relative importance must be maintained by the user.

At the other extreme are complex message filtering languages that can be used to automatically categorize, reroute, delete, and copy messages[1]. These languages are designed for computer experts or network administrators. In the hands of regular folks who happen to get a lot of messages, the complexity and power of these systems makes them potentially dangerous.

This paper describes Ishmail, a program designed for people who get a lot of email messages. Ishmail reduces information overload by automatically sorting messages into mailboxes and ordering mailboxes in order of importance. Ishmail minimizes the effort required to manipulate email by providing a user interface with customizable message and mailbox summaries that act as both views and interactive controls.

Messages are sorted by applying *message classification filters*, which identify messages that belong to a particular category or mailbox. Ishmail comes with several predefined message classification filters that users can parameterize. Users can also define their own filters and combine and modify filters with *and*, *or*, and *not*. Ishmail can apply filters to both incoming and outgoing messages.

Most email programs have only one type of alarm that alerts users about unread messages. Ishmail lets users define different types of alarms for different types of mailboxes. Mailboxes are checked to see if any alarms should be activated and then ordered by alarm status and priority and summarized in an interactive view.

Most email programs display two views of a mailbox: a message view, and a message header view that shows one-line summaries (i.e. headers) for each message in the mailbox. Ishmail adds two new views: a summary of mailboxes view (Fig. 1c.), and a log view, which records when each new message arrived and how it got classified (Fig. 1d.). One glance at Ishmail's summary and a user can tell which mailboxes require attention and why. Mailboxes that require attention are at the top of the list and marked with various alarm symbols. A click in the summary lets a user read any mailbox. Single keystrokes invoke other commands (e.g. edit mailbox definitions, show detailed information, reapply filters, etc.). Ishmail's user-interface is fully integrated with the GNU Emacs text editor [4] and the Emacs email program, Rmail.

We believe that Ishmail represents the proper compromise between a simple email program and a complex message filtering system. In particular, Ishmail has the look and feel of a familiar email program, but has extensions that let users control more than one mailbox at a time and treat each mailbox differently.

There are other email programs that help people classify messages [7, 9, 13]; however, as far as we know, Ishmail is unique in at least five respects. No other system supports: 1) mailbox classification and ordering, 2) different types of alarms on each mailbox, 3) the ability to read messages safely from multiple locations, 4) so many levels of archiving (the ability to make multiple copies of different parts of any mailbox), and 5) so many levels of customization (Ishmail exploits Emacs Lisp's extensibility and customization features such as variables, keymaps, and hooks).

Ishmail has been in continuous use by about a dozen people for more than a year and by several dozen people for several months. Many users have commented that now that they have Ishmail, they cannot imagine living without it. This paper describes the design and implementation decisions that we think make Ishmail useful, in particular, Ishmail's user-interface strategy and Ishmail's database and classification system. Generalizations about broader information filtering problems are also discussed.

User-Interface Strategy

Most email programs support two views of a mailbox: a message view, and a message header view. The same two views are also used for most news readers and bulletin-board systems. We believe that a user interface with both message and header views is useful because 1) it embodies a useful division between content and context and 2) it allows people to control content through context. The header view provides an overview of several messages (the context), but it hides the details of the individual messages (the content). The header view typically gives the context needed to tell which messages should be read first (e.g. which messages are unread and how old they are).

In Rmail, the message and header views are linked together by an intuitive user action. Selecting the header of any message displays it in the message view. When a user clicks the mouse button over a particular header they are using the context



Figure 1: Ishmail User Interface. Four views: a) the current message, b) headers of the current mailbox, c) summary of mailboxes, d) log. Here, the summary view indicates why three mailboxes require attention and the log view indicates recent activity.

to control the content. Information systems that let people control content by manipulating context are intuitive, easy to use, and give people power over their information.

Ishmail's user interface (shown in Fig. 1) provides the standard message view (Fig. 1a.) and header view (Fig. 1b.). In addition, Ishmail provides a higher level of context and control in the form of two additional views: a summary of mailboxes view (Fig. 1c.), and a log view (Fig. 1d.). For each mailbox, the mailboxes summary shows its alarm status, name, number of unread messages, total messages, date of oldest unread message, and current alarm thresholds. For each message, the Ishmail log shows its date and time of arrival, mailbox, sender (or recipient if you are the sender), and subject (or first line if there is no subject).

The mailboxes summary view provides the additional context needed to determine which mailboxes require attention. Selecting a mailbox in the mailboxes summary causes the header view to update, which in turn causes the message view to update. Reading a message causes the statistics in the mailboxes summary view to update, which in turn may cause mailboxes to be reclassified and possibly reordered to reflect their change in status.

One nice thing about the header view in a regular email program is that it provides temporal context. Since messages are ordered by arrival time, it is easy to identify the newest messages. A system that automatically classifies messages may

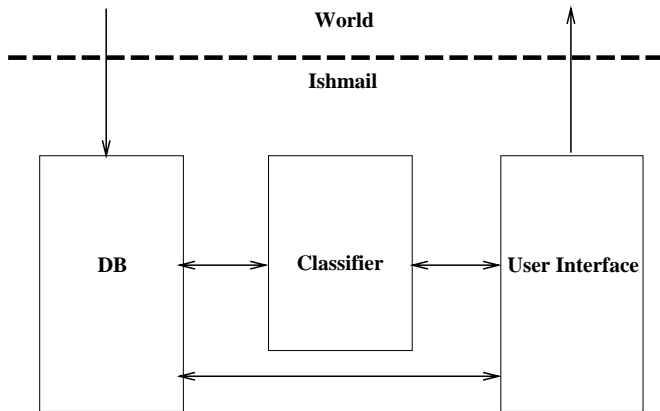


Figure 2: High-Level View of Ishmail's Functional Components

obscure these temporal relationships because the messages are scattered across many mailboxes. In Ishmail, the log view maintains a temporal record of message activity. The log records when each message arrived and also indicates how it was classified. Just one glance at the log and a user will be able to know where to find new messages. In addition, the log records whenever messages are moved between mailboxes (e.g. when resorting, sending, and copying).

The mailboxes summary and log views support two very different types of tasks. If there is a lot of unread email, the summary view identifies the important mailboxes that should be read first. The summary is useful when you have been away from your email and need to catch up quickly. There are other times, however, when you are waiting for a particular message to arrive. The log view supports this task nicely.

The Ishmail user-interface is intuitive and empowering. Each view gives the context needed to identify important information and the controls needed to access and manipulate it.

Design

Figure 2 diagrams Ishmail's design as three functional components: 1) a database to store messages and mailboxes, 2) a classifier to organize messages and mailboxes, and 3) a user-interface. Arrows indicate interactions between components.

The database provides mechanisms for the organization and manipulation of objects. Ishmail stores several types of objects; the most important is a *message*. Messages are organized into mailboxes and can be manipulated in various ways (e.g. created, deleted, moved, printed, etc.). Mailboxes are also objects. The database stores mailbox attributes, filters,

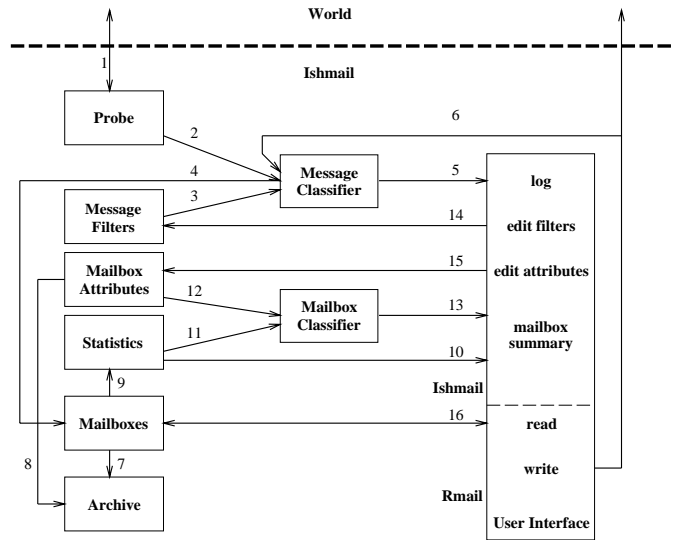


Figure 3: Detailed View of Ishmail's Functional Components

and statistics. Mailboxes can also be organized (by alarm status, priority, etc.) and manipulated (added, deleted, etc.).

The classifier engine has a language that allows users to describe what parts of an object are important and how objects should be grouped together. Ishmail is mainly concerned with classifying messages and mailboxes (i.e. groups of messages). Although it could, the Ishmail classifier does not currently determine which parts of an object are important enough to display (e.g. the body or header of a message, or the unread messages in a mailbox). It also does not currently compare filters to help users determine potential conflicts (but see **Future Extensions** below).

The user interface presents views that reflect the organization of a user's data. Ishmail reflects message classification in the log view and mailbox statistics in the summary view. The user interface also provides controls that allow a user to manipulate data objects, construct new objects, and present objects to the outside world. It also users to edit mailbox attributes and message filters.

It is useful to think about the three main functional components in a bit more detail. For example, in Ishmail, the abstract database component is really composed of multiple databases and a probe, a subcomponent that checks for the arrival of new messages from the outside world. Similarly, the classifier is really composed of a message classifier and a mailbox classifier. Figure 3 illustrates these subcomponents. Numbered arrows indicate subcomponent interactions. Each interaction is briefly described below. Implementation details are found in the following sections.

When a new message arrives, the probe (1) moves it out of the

spool area, the public part of the file system where everyone else's messages are stored, and invokes the message classifier (2). The message classifier applies the message filters (3). The classified message is appended to the appropriate mailbox (4) and the log view is updated (5). The message classifier also applies filters to outgoing email (6). Whenever a mailbox is updated, an associated archive mailbox may also need to be updated (7). Archiving behavior is determined by the settings of the mailbox attributes (8).

After classifying messages, various mailbox statistics must be updated (9). The updated statistics are displayed in the mailboxes summary (10). As mailbox statistics change, mailboxes must be re-classified (11). Mailbox re-classification is governed by the mailbox statistics and attributes, particularly alarm thresholds and priorities (12). After mailbox classification, mailboxes are listed in order of importance in the mailboxes summary (13).

The Ishmail user-interface lets users edit their message filters (14) and mailbox attributes (15). In addition, when an unread message is displayed, it is marked as *read* possibly changing the state of its mailbox (16).

Implementation

When we started working on Ishmail we wanted a message management system that would serve as a platform for related research in text classification, data visualization, learning, and user-interfaces for managing information. It was also important for us to implement Ishmail quickly. We were drowning in a flood of email and needed a prototype that would get us back on our feet. Daily use of the prototype has allowed us to understand better the issues related to managing large amounts of email.

We felt strongly that the prototype should look and feel like a familiar email program. People are comfortable using email programs to read, compose, send, and save messages. It would be ideal if our system could operate in the background and seamlessly add capabilities to everyone's favorite email program. Unfortunately, most email programs are not designed to tell other programs when messages are read, saved, or sent. We needed to track these operations to maintain accurate mailbox statistics. Since we did not want to write yet another email program, we looked for an existing email program that we could modify.

Ishmail is implemented in GNU Emacs, a public-domain text editor. Emacs is extremely customizable, primarily because it supports an interpreted extension language, Emacs Lisp [8]. An extension is a collection of Lisp functions and variables that are loaded by Emacs. One example of an Emacs extension is Rmail, the Emacs email program.

Ishmail is implemented as an extension to Rmail. Ishmail calls several Rmail functions and accesses a few Rmail variables. In addition, Ishmail utilizes Rmail *hooks*, and overrides Rmail *keymaps*. Hooks are lists of functions called at critical points in a program. A keymap is a mapping from keystrokes to functions. We found Emacs to be a wonderful prototyping environment; however, like most heavily used and frequently modified programs, Rmail has evolved to a point where the

interactions between its components are abundant and confusing to the uninitiated. Still, were it not for the general level of extensibility of Emacs, we would never have been able to build such a powerful, customizable, and useful prototype in such a short amount of time (we started working on Ishmail in the summer of 1994 and there were several loyal users by the fall).

Implementation: Probe

Most email programs retrieve messages from the public spool area only on demand. By contrast, Ishmail's probe is always checking for and retrieving new messages. The probe is implemented with the Emacs `alarm` function and a C program, `movemail`, which comes with GNU Emacs. At regular intervals, the probe checks for new messages by testing the size of the user's email spool file. If the spool file is not empty, the probe uses `movemail` to move it to a temporary mailbox. The `movemail` program uses an elaborate protocol that tries to avoid file corruption even if new messages arrive while the spool file is being moved (it is, however, possible to get multiple copies of messages and null characters inserted into messages, although these problems seem to happen rarely).

Several questions arise because Ishmail is always retrieving new messages. For example, how often should it check? If it checks too often, the responsiveness of the operating system may be compromised because constant polling wastes resources. If it does not check often enough, a user will not get messages in a timely fashion. Furthermore, since Ishmail does not respond to input while it is classifying messages, there are shorter delays if it classifies messages as they arrive instead of in large batches.

Although the delay for classifying a particular message is typically short, it can be frustrating to be interrupted when composing messages or otherwise editing text. Therefore, Ishmail will not check for new messages under these circumstances.

The most drastic way to prevent Ishmail from checking for new messages is to quit the program. Quitting Ishmail might result in a large accumulation of unread messages. Therefore, it makes sense to leave Ishmail running all the time. This raises a problem that is common to all email programs: what do you do when you want to read messages, but your email program is running somewhere else? The first email program may have unsaved changes. It would be disastrous to let the second email program make changes of its own. For Ishmail this problem is particularly acute. Two concurrent Ishmail processes would compete for the same messages, eventually leading to an inconsistent database.

Ishmail solves this problem by implementing a process for transferring control. When Ishmail starts, it detects if another Ishmail process is already running by looking for a special *control* file that the first Ishmail will have created. If the control file exists, the second Ishmail will verify that the user wants to transfer control. If so, the second Ishmail creates a *transfer* file, which the first Ishmail checks for periodically. When it detects the transfer file, it saves its changes, deletes its control file, and quits. The second Ishmail waits until the control file is deleted before creating its own control file

and proceeding to check for new messages. If the second Ishmail waits too long, it decides that the first Ishmail has died, and asks the user if it should proceed under that assumption. Communicating via files is a reliable mechanism for maintaining data integrity in a network of heterogeneous machines.

Implementation: Databases

The abstract database component is really composed of multiple databases. The implementation of the databases for storing mailboxes, mailbox attributes, mailbox archives, and mailbox statistics is described below. The database for storing message classification filters is described in **Implementation: Classifying Messages**, below.

Ishmail implements each mailbox as a UNIX file. Since Ishmail is an extension to Rmail, we use the Rmail mailbox format, BABYL. BABYL is an ASCII file format that uses control characters to mark message boundaries. BABYL also introduces some extra message fields that store state information about each message (e.g. if the message has been read). We have experienced some problems with the BABYL format. In particular, PostScript files that include control characters can occasionally confuse Rmail.

Mailbox attributes are stored in an ASCII file along with a user's message classification filters. Attributes take the form of name/value pairs. Users can override the default values of each attribute when they define their mailboxes to customize how Ishmail organizes their email. For example, the `mail-flow` attribute is used to direct Ishmail to classify messages when they arrive (the default), when they are sent, or both. There are other attributes that determine a mailbox's archiving behavior and priority (see **Implementation: Classifying Mailboxes** below). Users can also define their own mailbox attributes.

An Ishmail archive is a copy of an Ishmail mailbox. Messages may be archived automatically or manually by setting the appropriate mailbox attribute. Archiving has two major purposes: 1) to save copies of potentially important messages and 2) to keep mailboxes small. Keeping mailboxes small is a good idea because it is easier to find messages in a small mailbox (also, we have noticed that Rmail tends to slow down when mailboxes store more than a few hundred messages). Whenever a message is appended to a mailbox that archives automatically, a copy of the message is stored in the archive. Automatic archiving is useful for mailboxes that often have important messages. It is safe to delete all but the most crucial messages from these mailboxes because there is a backup copy in the archive. By contrast, manual archiving is useful for mailboxes that occasionally have important messages. With two keystrokes the important messages can be copied into the archive and then the entire mailbox can be safely deleted.

A mailbox can have any number of archives associated with it. Archives are usually grouped by the year, month, week, or day of their messages. Other groupings are also possible. The most frequent types of archives are yearly and monthly. Automatically archiving by year is appropriate for small mailboxes that usually get important messages. Auto-

matically archiving by both year and month will make two copies of messages, one in a large yearly archive, and the other in a smaller monthly archive. The advantage of two archives is that if you need to refer to a message quickly and you remember the month it was sent, you will be able to find it faster in the small archive than you would in the large archive (particularly near the end of the year). Every few months you can manually remove the older monthly archives. If a mailbox receives many messages, also archiving by week may make sense.

Ishmail maintains private state information about mailboxes (the total number of read and unread messages, the date of the oldest unread message, etc.). Calculating these statistics can be slow if there are several large mailboxes. In order to minimize delays, the statistics are updated incrementally as mailboxes change state. The statistics are stored in a database, implemented as an ASCII file, which Ishmail reads when it starts. In this way, Ishmail can start quickly without having to read the dates and attributes of old messages. Ishmail will recalculate the statistics automatically if it detects that they do not accurately reflect the state of the mailboxes.

Implementation: User Interface

Ishmail's user interface is built on top of Rmail's. Rmail supports the standard message operations such as read, save, delete, compose, and send. In addition, Ishmail provides new commands to print messages, delete messages that meet certain criteria, and other useful enhancements.

The Ishmail user interface also adds two new views to Rmail: 1) a summary of mailboxes, and 2) a log. These views are implemented as Emacs buffers with special keymaps. In both views there are keystrokes that will run Rmail on a mailbox, clear the log, quit Ishmail, and print a keymap description. In the summary view there are also keystrokes that will set alarm thresholds, apply classification filters to a mailbox, and display mailbox information.

In addition, Ishmail allows users to edit their mailbox definitions, including their classification filters and mailbox attributes. Mailbox definitions are edited with normal Emacs text-editing commands in a special Emacs buffer that has a few extra key bindings. There are key bindings that will check the syntax of filters and attributes, abort editing, and show online documentation that describes the existing filters and attributes and legal values for their parameters.

The user interface for defining mailbox classification filters is currently the weakest part of Ishmail's user interface. Expressions in Emacs Lisp must be written to specify each filter. This usually is not too hard because it is easy to copy and modify filters from the examples in the documentation or from elsewhere in the rules file. Nevertheless, easier interfaces for specifying classification filters exist in other email systems [14] and are being developed for Ishmail (see **Future Extensions** below).

Implementation: Classifying Messages

The message classifier applies message classification filters to each message presented to it. In Ishmail, each filter is associated with a single mailbox. The message is moved into

the first mailbox with a matching filter. If no filters match, then the message is moved into the user's default mailbox.

When messages are classified, they are grouped together with other messages that are in some way similar. Since messages consist of strings of ASCII-encoded characters, most measures of message similarity require string comparisons. So when we think of two messages as being similar, we usually mean that they have one or more strings in common. In particular, since message headers have several different fields, messages may be similar if they have the same string in the same field.

For example, it may be desirable to group together the messages from a particular person, organization, or mailing list (e.g. the `hiphop` mailing list). These messages can be easily identified because they will all have the string `"hiphop"` in the `"To:"` or `"Cc:"` field. A message classification filter would only have to look at each message's `"To:"` and `"Cc:"` fields to see if it found the string `"hiphop"`. This sort of classification is so common that Ishmail has a pre-defined filter that can be parameterized for this purpose:

```
(imail-in-any-field "hiphop" "To:" "Cc:")
```

With Ishmail, it is possible to create a more specific filter by combining two primitive filters with the conjunction `imail-and`. This filter will group messages that are both to the `hiphop` mailing list *and* about Public Enemy:

```
(imail-and
  (imail-in-any-field
    "hiphop" "To:" "Cc:")
  (imail-in-any-field
    "Public Enemy" "Subject:"))
```

What about collecting messages that are both sent to and received from the same person? Here is an Ishmail filter for use with a mailbox that accepts both incoming and outgoing messages:

```
(imail-in-any-field
  "elvis" "From" "To:" "Cc:")
```

This filter might not be quite right because it will match messages to `elvis` that did not come from you. Here is a more specific filter that will match both messages *from* `elvis` and messages from you *to* `elvis`:

```
(imail-or
  (imail-in-any-field
    "elvis"
    "From:")
  (imail-and
    (imail-in-any-field
      "elvis"
      "To:")
    (imail-in-any-field
      "you"
      "From:")))
```

The more specific filter is a combination of three primitive filters. In addition to combining filters with `imail-and` and `imail-or`, you can also modify them with `imail-false` to create arbitrarily complex filters.

There are several other Ishmail filters that can be used to classify messages. One filter looks for any number of strings in any number of message fields. Others look for a string in any part of the message header or in any part of the message body. There is also a filter that will look for a string anywhere in the entire message. And there are filters that take Emacs regular expressions as arguments, instead of strings. Since regular expressions can match many different strings, these filters are very general, but run a bit slower than the ones that take strings as arguments.

A preliminary study of how people use Ishmail filters revealed that there are roughly three classes of filters: rarely used, commonly used, and abundant. The abundant filters are used more than twice as much as the commonly used ones. The abundant filters, like `imail-in-any-field`, look for strings that may appear in some number of fields. The commonly used filters included `imail-and` and `imail-or`. The remaining filters were used rarely. For example, there was only one usage of the filter `imail-not` and no one used regular expressions. These results are significant because they suggest that most email message filtering tasks can be accomplished without elaborate and complex tools such as negation and regular expressions (see **Discussion: General Information Filtering** below).

Implementation: Classifying Mailboxes

Mailboxes must be classified whenever Ishmail starts, the mailbox statistics change, or the mailbox definitions are edited. In Ishmail, the purpose of classifying mailboxes is to be able to indicate which mailboxes are important and why. Mailboxes are displayed in order of importance. Importance is determined by status, priority, and definition order.

First mailboxes are classified by status. There are four status classes: 1) at least one active alarm, 2) some unread messages, 3) no unread messages, 4) no messages. Mailboxes with active alarms are ordered before those with unread messages, which are before those with no unread messages, which are before those with no messages. Within each status class, mailboxes are further ordered by their priority attribute. The priority attribute can take one of three values: `high`, `medium`, or `low`. Finally, if mailboxes share the same status class and priority, they are ordered as their filters appear in the filter database.

Implementation: Extensibility and Customizability

Ishmail is quite extensible and customizable. There are about forty user-definable variables. It is also possible for users to define their own primitive filters and to specify new mailbox attributes. By combining new filters and attributes with the various hooks that Rmail and Ishmail provide, it is possible to add powerful new functionality to Ishmail. We discuss some possibilities below in **Future Extensions**.

Discussion: General Information Filtering

Message filtering is an instance of the general problem of information filtering. Ishmail's design reflects the problem's three parts: 1) database, 2) classifier, and 3) user interface.

We believe that the right way to implement a messaging system is with a real database—probably an object-oriented

database. Each message is an object. Messages are initially labeled by message classification filters. Mailboxes are implemented as views of objects with particular labels. There is at least one email program that comes close to this approach, MH, but it models the database with a file system (each message is a file; mailboxes are directories of symbolic links to files) [12]. Emulating a database may work for small amounts of data, but for large amounts of data a real database is needed, and all filtering systems eventually have to deal with large amounts of data if they are to be useful.

Using a real database would solve many of the problems that we encountered. For example, Ishmail does not classify a message into more than one mailbox; the classifier stops at the first mailbox with a filter that accepts the message. It is certainly possible to have the classifier copy messages into each mailbox with an accepting filter, but this would lead to a preponderance of duplicate messages. Allowing duplicate messages requires an efficient method for deleting a message together with its duplicates. In a real database, objects can occur in multiple views, so there is no need for multiple copies.

Many other problems would also vanish (e.g. no need for a mailbox format like BABYL if each message is its own object, no size restrictions on mailboxes, no problem supporting multi-media objects, etc.) The reason we did not implement Ishmail with an object-oriented database is because we needed a quick prototype that would be maximally flexible and easily extensible.

As we have noted before, it is important to be able to organize and classify views as well as objects. Even object-oriented databases are not designed to organize views automatically. It is the job of a classifier to automatically organize views into hierarchies and place objects in the correct position in the hierarchy. Recall our earlier examples of a filter that would group messages from the `hiphop` mailing list:

```
(imail-in-any-field "hiphop" "To:" "Cc:")
```

and a filter that would group messages from the `hiphop` mailing list about Public Enemy:

```
(imail-and
  (imail-in-any-field
    "hiphop" "To:" "Cc:")
  (imail-in-any-field
    "Public Enemy" "Subject:"))
```

A system that organizes filters hierarchically would notice that the first filter *subsumes* the second. That is, any message that meets the criteria of the second filter will also meet the criteria of the first. We call the second filter a *child* of the first. In Ishmail, a filter describes the kind of messages that a mailbox contains. Therefore, we can say that the `Public Enemy` mailbox is a child of the `hiphop` mailbox.

In a system that recognizes subsumption relationships between mailboxes, it becomes easy and efficient to ask for all messages that meet the criteria for a particular mailbox but do not meet the criteria for any of the mailbox's children (e.g. messages to `hiphop` but not about Public Enemy). It is important to be able to ask this kind of question when messages

can be classified into several different mailboxes at the same time.

The problem of subsumption has been studied extensively. It is well known that the general problem of organizing filters is intractable, particularly with systems that allow *and*, *or*, and *not* [2, 3]. It is therefore important to decide the power of the classification language that will be allowed. The language must be flexible enough to classify the objects we care about but not so powerful that it is impossible to detect conflicts and subsumption relationships. We believe that for filtering messages, it is possible to strike this balance because most message filters do simple string comparisons and use only *and* and *or*.

In addition to classifying objects and views, it is important to be able to describe the parts of objects and views that are relevant in particular situations. For example, when you receive a message informing you that some email could not be delivered, you will only be interested in viewing the parts of the message that describe what the problem is. Similarly, for the header view of a mailbox that collects these kinds of messages, you may not want to see the subject field because that usually does not contain any useful information.

Most would agree that the process of classifying objects is the job of a classifier. Many consider the process of determining when parts of an object are relevant to be the job of a user interface designer. The notion that a designer can always know what parts of a message are going to be important is the same incorrect notion that a designer can always know when a mailbox requires attention. People should retain the power to decide how and when to view their data.

We believe that classifying objects, groups of objects, and parts of objects are all instances of information filtering. Just as you use a classifier to decide when you want to see a particular object or group of objects, you can also use a classifier to decide when you want to see particular parts of an object. The classification language provides a powerful mechanism that can serve *any* process that can work on your behalf, not just the user interface (e.g. an agent that places announcements for talks and meetings in your calendar program).

It is clear that the user interface must provide a way to view and manipulate objects in a database. With a powerful classification language, it becomes just as important for a user interface to provide a way to view and manipulate the language. No matter how powerful, complicated and cryptic the underlying classification language is, people must be able to easily construct expressions in the language and feel confident that their expressions accurately reflect their wishes.

Although we believe that Ishmail is intuitive as well as powerful, its user interface to the classification language could be easier to use. The interface to the language makes it easy to create and modify filters. It could be improved if it also made it easier to visualize interactions between filters (e.g. subsumption relationships and potential conflicts).

It is important to realize that the message filtering problem is an instance of a more general information filtering problem. Lessons learned from Ishmail should be applicable to this

more general problem. Insights into the more general problem should be brought to bear when designing other kinds of information management systems.

Future Extensions

Several extensions to Ishmail are in various stages of progress. For example, we have written a prototype graphical alerter in Tcl/Tk [11]. The graphical alerter displays animating icons for each mailbox with active alarms. Selecting an icon invokes Ishmail on the corresponding mailbox.

One of our colleagues is planning to adapt his learning algorithms to automatically determine message classification filters from existing mailboxes [6]. Another colleague is planning to extend the Rmail header view to provide more control over mailboxes that contain multi-threaded conversations.

Two other colleagues are designing a system to help people define classification filters and detect interactions between them. This system could eventually replace Ishmail's textual specification of filters, but it may also be useful for related problems in image and audio classification [17].

We believe that information management systems should be maximally flexible and easily extensible. We are encouraged by these projects because they suggest that Ishmail is not only useful, but is general and extensible enough to be used as a platform for further research.

Conclusion

We have identified information overload as a critical problem for people who get a lot of electronic mail. We have described Ishmail, a customizable email system that addresses the problem of information overload. We have summarized related work and indicated that Ishmail is unique in that it not only sorts messages into mailboxes, but it orders mailboxes by a combination of user-specified priorities and alarms. We have also identified three other features that we believe are unique to Ishmail: 1) transfer of control, 2) multiple levels of archiving, and 3) multiple levels of customization.

We have shown how Ishmail's user-interface allows users to manipulate messages and mailboxes by controlling content through context. Further, we have shown that Ishmail not only gives users the context needed to determine when mailboxes are important, it also gives control over the details of that context.

We have diagrammed Ishmail's design in terms of its functional components and their interactions. We have also described our implementation of each component. In addition, we have shown that message filtering is an instance of the general problem of information filtering. We have presented an argument that information filtering should be addressed with a three-part strategy of 1) database, 2) classifier, and 3) user interface and we have indicated that this three-part strategy is echoed by the architecture of Ishmail's functional components. Finally, we have summarized future extensions planned by ourselves and our colleagues.

REFERENCES

1. *FLAMES: Filtering Language for the Andrew MESSage System*.
2. Alex Borgida, Ronald J. Brachman, Deborah L. McGuinness, and Lori Alperin Resnick. CLASSIC: A structural data model for objects. In *Proceedings of the 1989 ACM SIGMOD International Conference on Management of Data*, pages 59–67, 6 1989.
3. Ronald J. Brachman, Deborah L. McGuinness, Peter F. Patel-Schneider, Lori Alperin Resnick, and Alex Borgida. Living with CLASSIC: When and how to use a KL-ONE-like language. In John Sowa, editor, *Principles of Semantic Networks: Explorations in the representation of knowledge*. Morgan-Kaufmann, San Mateo, California, 1991.
4. Debra Cameron and Bill Rosenblatt. *Learning GNU Emacs*. O'Reilly & Associates, Inc., 1991.
5. Maria Capucciati, Patrick Curran, Kimberly Donner O'Brien, and Annette Wagner. Neither rain, nor sleet, nor gloom of night: Adventures in electronic mail. In *Human Factors in Computing Systems CHI '95 Conference Proceedings*, pages 553–557, Denver, Colorado, 5 1995.
6. William W. Cohen. Fast effective rule induction. In *Machine Learning: Proceedings of the Twelfth International Conference*, Lake Tahoe, California, 1995. Morgan Kaufmann.
7. David Goldberg, David Nichols, Brian M. Oki, and Douglas Terry. Using collaborative filtering to weave an information tapestry. *Communications of the ACM*, 35(12):61–70, 1992.
8. Bil Lewis, Dan LaLiberte, and Richard Stallman. *GNU Emacs Lisp Reference Manual*. Free Software Foundation, Inc., second edition, 8 1993.
9. Thomas W. Malone, Kum-Yew Lai, and Christopher Fry. Experiments with oval: A radically tailorable tool for cooperative work. *ACM Transactions on Information Systems*, 13(2):177–205, 4 1995.
10. Hanna Nelson. *The Z-Mail Handbook*. O'Reilly & Associates, Inc., 1991.
11. John K. Ousterhout. *Tcl and the Tk Toolkit*. Addison-Wesley, 1994.
12. Jerry D. Peek. *MH & xmh: E-mail for Users and Programmers*. O'Reilly & Associates, Inc., 1990.
13. Steven Pollock. A rule-based message filtering system. *ACM Transactions of Office Automation Systems*, 6(3):232–254, 1988.
14. Qualcomm Inc. *EUDORA Macintosh User Manual*, 9 1993.
15. J. Rosenberg, C. F. Everhart, and N. S. Borenstein. An overview of the andrew message system. In *Proceedings SIGCOMM '87 Workshop on Frontiers in Computer Communications Technology*, pages 99–108, Stowe, Vermont, 8 1987.
16. Kurt Shoens and Craig Leres. *Unix User's Manual Supplementary Documents: 4.3 Berkeley Software Distribution*, chapter USD7. USENIX Association, 8 1986. Mail Reference Manual.
17. L. G. Terveen and L. Murray. Helping users program their personal agents. in submission to CHI '96.