MAKING MAIL FRIENDLIER:
ADDING MACINTOSH FEATURES AND MULTIMEDIA DOCUMENTS TO UNIX MAIL

John R. Meier

Technical Report PCS-TR86-135
Making Mail Friendlier:
Adding Macintosh features
and multimedia documents to
UNIX Mail

John R. Meier
Senior Honors Thesis
Computer Science Department
Dartmouth College
May 27, 1986

Abstract

This paper describes a Macintosh application which acts as a front-end to Unix Mail. Features of the Macintosh interface such as icons, menus, and windows replace the command driven interface. Complicated editing commands are replaced with mouse selection and cut, copy, and paste. Messages can be composed of text, pictures, and any Macintosh file, because they are encoded into plain text, sent through the mail system, and then unencoded by the receiving end. The designs of the mail server and communications interface are such that mail servers and communications other than Unix mail and a serial line may be easily implemented.
Table of Contents

Abstract

Table of Contents

1. Summary

2. Goals
   A Macintosh User Interface to Mail
   Multimedia Documents
   Minor Goals
   Non-Goals

3. Message Structure
   Background
   Design
   Implementation
   Results and comments

4. User Interface
   Background
   Design
   Implementation
   Results and comments

5. Communications Interface
   Background
   Mail Server Interface
   Stream File Interface

6. Wrapup
   Overall Results
   Future Work

7. Annotated Bibliography
   Message Structure
   User Interface
   Communications Interface
In this thesis on a Macintosh interface to electronic mail, the first section discusses goals of the project. Sections 2-4 discuss the message structure, user interface, and communications interfaces of the mail system. Section 5 wraps up the thesis, and a bibliography is included in Section 6.

The major goals of this thesis are described in Section 1. One major goal is to add a Macintosh interface to Unix mail. The mouse commands of the Macintosh interface will be easier to learn than obscure commands for a Unix editor and Unix mail. The Macintosh interface will already be familiar, because as a Dartmouth student, the user will have already learned to type papers on a Macintosh. The other major goal is to add multimedia documents to Unix mail in order to permit the transfer of any of the complex Macintosh documents. Minor goals are to keep the mail server and communications interfaces flexible, retain mail transfer to text-only users, and allow for the adoption of new document standards. Non-issues for this thesis are creating a complex editor, knowing the internal operation of the mail server, and finding the optimal byte to text encoding.

In section 2, the message structure used in the mail system is explained. Messages contain an envelope, a text comment, a picture, and an attachment. The envelope contains addressing information, and the text comment can contain any plain text. The picture can hold any Macintosh picture, and the attachment can be any Macintosh file.

The user interface is described in section 3. The mail desktop is displayed as a Mailbox window containing folders. These folders are opened to show messages. Opening a message creates a window displaying the contents of the message. The message can be edited using cut, copy, and paste. Sending the message is another command. The structure of the implementation is to send an object to an operation (send folder1 to draw) in order to perform the operation. This structure is general and is easily extended to new classes of objects.

Section 4 describes the communications interfaces necessary to communicate with the mail server. Layering is used to maximize flexibility. The first interface is between the communications link and a stream file. The next layer uses the stream file to create an interface between the Macintosh, and the mail server.

Section 5 discusses final results of the mail interface and of future directions in which to work. The mail interface was successful. Not all features were implemented, but the two major goals were accomplished, and it was shown that such an interface could replace the existing procedure for reading mail. Future directions for mail will probably be dependant upon tightly connecting disparate workstations between themselves and other computing resources.

The bibliography is in the last section, number 6. This bibliography is divided into three parts corresponding to sections on message structure, user interface, and communications interface, respectively. A short summary of important ideas in each source follows each entry in the bibliography.
Major Goals

The two primary aims of this thesis are to add a Macintosh user interface and multimedia documents to Unix mail. There are also minor goals in this thesis, and items which are specifically not goals of this thesis.

Macintosh Mail Interface

There are two reasons why a Macintosh user interface to Unix mail is needed. A Macintosh interface is an obvious choice at Dartmouth, because the college has standardized on the Macintosh. A secondary reason is that there are problems with the Unix mail interface which makes it hard for new users.

The most important point is that the user had already learned how to use the Macintosh to write his papers. The college has encouraged all students to purchase a Macintosh by installing a Macintosh compatible network in all the dorms. Dartmouth has also made an agreement with Apple to sell the Macintoshes at a heavy discount. The response has been that the overwhelming majority of students have purchased a Macintosh. Once the Macintosh made its mark, students discovered word processing and now almost all of the papers, labs, and assignments are done on the Macintosh. This means that the student understands mouse editing, windows, and menus, and so will not have to learn or remember a new process to use mail.

Unix mail and Unix in general are not the friendliest environments for new users. Commands are not visible and must be memorized (if they can be found somewhere). Once found, commands are terse and have cryptic codes. Modes are frequent and confusing. The biggest mode, between editing and mail, requires the user also to learn how to use a complicated editor. In many cases, the problems of using Unix are compounded because the user is using a terminal emulator. The user must startup the communications program, logon to the host from the network, login to his individual account, and then give the mail command just to check his mail.

Multimedia Messages

Multimedia messages are needed for Unix mail, because plain text is no longer used on the Macintosh. Papers now have different fonts, and variations in style. Pictures, charts, and graphs are included in these papers. Stripping non-text from these files causes too much information to be lost. It is possible to convert these documents into a different format, upload them with a terminal emulator into a message, and then have the receiver do the reverse process. All of these steps need to be done automatically, and will be done by adding multimedia messages to Unix mail.

Minor Goals

There are several minor goals for this project. The first minor goal is to provide an easy transition from text only mail to this new multimedia system. Being able to still communicate in the same way as before and then to gradually increase use of
multimedia mail will raise the chances of its acceptance and lower initial resistance. Another minor goal is to easily make the transition to using a new document standard. Eventually, a document standard will be accepted. When this document standard is accepted, this mail system must be changed to recognize this standard. The third minor goal is to keep the interfaces between the server and the communications link clean. This goal is important because Dartmouth is a campus with many different possible mail hosts. There also are more than one kind of communications link available. Keeping these interfaces clean will allow this mail system to be used as an interface with any combination of server and communications link.

Non-Goals

As this project is of finite size, there are certain tasks which I am choosing not to be part of this thesis. The first and largest of these is that I don't want to create a new editor. Many document editors already exist, and most also took an extremely long time to produce. Another specific item which I am choosing to ignore is the internal operation of the mail server. Though issues such as routing, delivery, addressing, and file storage are interesting, they will not be a part of this thesis. Because of limitations in the chosen communications interface and mail server, bytes must be translated into printable text characters before being sent to the server. The method of this encoding is unimportant to me. I will use the simplest method, and any methods which pack data more densely will only improve the response of my mail system.
Message Structure

Messages are the central object in mail. They are sent, received, created, edited, and viewed. A multimedia message structure is one of the major goals of this thesis. Because messages are so important, the structure of these messages is also very important. In the beginning of this section, previous ideas for message structure are discussed. Next, the structure of messages in this mail system is explained. The details of implementation follow the message structure explanation, and results and comments are at the end of this section.

Message Background

Message Standards

IBM DOA/DCA [Parks], X.400 [Horak], USC Multimedia [Postel], RFC 822 [Crocker], and NBS [Deutsch] are all different proposals for complex message standards. One common theme is to separate message layout from message content. Page margins, and line spacing are layout objects, and chapters and paragraphs are content objects. Another common idea is to create a hierarchy in the message defining objects as being composed of smaller objects. The USC Multimedia format is especially interesting because a list of objects can be of simultaneous, sequential, or unordered type. This type affects how the parts of the message are displayed. No other standard allows anything but sequential as a possibility. RFC 822 is the only standard which has been put into use by the academic community. RFC 822 completely defines the possibilities for header fields, yet leaves specific types of message content undefined.

MacMail Messages

MacMail [Davis and Hourvitz] messages have a complicated header, a text comment, and a list of attachments. The header is an arbitrarily long list of fields. These fields could be used to satisfy the X.400 message standard or to provide a kind of structured message. A phone message with fields for "Call From", "When", "Action Required", and "Answered by Whom" is an example of a structured message. "Simple" messages are limited in size of the text comment and the header is fixed. Pasting of pictures is not supported forcing any non-text items to be attachments. Reading any non-text files requires the user to exist from mail and start the appropriate tool.

Tree Structured Messages

NLS [Feiner] and Thinktank [Think Technologies] structure documents as a large tree with levels of detail at different depths in the tree. Links to more detail about a section can be found in the node's children. Settings define the visible default depth, or the user may choose to see more detail at any time. Display of this document is different because display depends on both the content of the document and the actions of the user.
Active Messages

Active messages have data, but they also have instructions that can perform actions [Vittal]. Other messages have a format that structures the data, but the message is entirely data. When the message is "read", the instructions for a message are performed. As an example, let's send a questionnaire to all the members of a calculus class. When each student reads the message, the code in the questionnaire is executed, asking questions and getting responses. After the final question, the active message instructions put the results into a reply and mails it back to us.

The largest advantage of active messages is that new active messages can be added easily. The only requirement is that everyone has an interpreter for these active messages. Choosing the language that these messages are written in is difficult. The language must protect the reader from hostile messages. (e.g. upon reading this message, all your private files are sent to me) A more difficult case is to prevent messages that go into an infinite loop. The halting problem states that you cannot detect these infinite loops, yet you would like to give the user the power of those same languages. In practice, these infinite loops can be avoided by careful testing or by limitations in the active message language. Another use for an active message is to aid in sorting and sifting mail. An active message could be opened that looks at all the messages in the inbox and opens some important ones, and files others appropriately. I think that active messages are a good idea, yet I have chosen not to use them because developing a language and interpreter would be a thesis in itself.

Smalltalk

Smalltalk is a generalization of the active message concept [Goldberg and Robson]. Instead of having every message contain a procedure, every message is in a class, and each class has procedures. The procedures don't have to be just for "reading", but are for all actions. To display a message, you ask the message to display, and it uses the display procedure for the class of that object. There could be objects of class picture, text, and textPictureList. Each of these classes would have different display procedures.

Smalltalk is a single user environment, and it only looks in its own environment for procedure for different classes. For use in mail, Smalltalk would have to be extended to retrieve class procedures for classes which it lacks.

Inset Drivers

Inset drivers are part of the Andrew workstation design at CMU [Hansen] [Morris et al]. Like Smalltalk, events for each inset are sent to a driver for the class of that inset. The difference is that insets are a campus-wide idea. There will be standard types of insets (TEXT, PICTURE, etc.). There will be insets developed for small groups (e.g. Molecular Structure), and one can also make his own insets. All insets will be required to handle a certain set of operations. Some of these operations (like mouse click) are general enough that any type of inset could be built. An example is an inset of type
Message Structure

PICTURE. First, we install a PICTURE driver with network-wide access. When mail tries to display a message with a picture, the picture driver is retrieved over the network, and the picture driver's display routine draws the picture. When a mouse clicks in the picture inset, the picture driver's mouse routine is called and perhaps some editing is performed.

This structure is great. It eliminates repeated work, and allows good generality and extensibility. Unfortunately, a large amount of memory would be needed to hold all the drivers, and accessing these drivers over Dartmouth's network is too slow. The architecture of the Macintosh software would have to be changed to think of events in a window going to the driver instead of the application. Additionally, all applications should be written around this structure, or they won't take advantage of the drivers.

Message Design

Every message consists of four parts: an envelope, a text comment, a picture, and an attachment. Of these parts, only the envelope is required. The envelope contains header information. The text comment is a piece of plain text, the picture is any Macintosh picture, and the attachment can a file of any type.

![Message Example]

Figure 3-1: A message containing only an envelope

Envelopes

The envelope's main purpose is to contain addressing information. It must have any information needed by the mail server to deliver the message. In addition, it may have fields such as "date delivered" and "return address", or additional fields created by the user or mail server. Additional fields are useful for specific applications, and are required to be compatible with RFC 822.

An important point is that only information in the envelope is visible to the mail server. Other parts of the message are sent verbatim from the sender to the receiver. The analogy of the US Postal Service fits well with this design. When you mail a letter at
the post office, the address on the envelope is referenced, the stamp is checked, and a postmark is added, yet the contents inside the envelope are never seen.

Figure 3-2: A message with text comment and envelope

The Text Comment

The text comment is a piece of plain unformatted text. The main reason for a text comment is for creating a short, simple message. Most mail messages are very brief and it must be possible to quickly create and send these messages [Bruder] [Tapscott]. Most existing mail systems allow messages of only text, so their messages are placed in this text comment. Another use for the text comment is to quickly add notes or revision ideas to a previous message before returning or forwarding it. Text is also one of the data structures that Macintosh applications should be able to accept from the clipboard.
Figure 3-3: A message with picture and envelope

The Picture

Every message may also contain a picture. A Macintosh "picture" is the other data structure that applications should be able to accept from the clipboard. It is important to support the picture data type, because you are guaranteed that all Macintosh users will be able to display this picture. If the same picture were saved as an attachment in MacDraw format, the recipient would need to possess the MacDraw program to read your message. It is unreasonable to assume that all users own all programs, because software packages are sold (for $$$!) individually. This is in contrast to the Xerox Star and the Apple Lisa where all the software was bundled with the machine, and you were guaranteed that the receiver would possess the application needed to read your message.

![Message with picture and envelope](image)

Figure 3-4: A message with attachment, text comment, and envelope

The Attachment

A message has an optional attachment which is a file of arbitrary type. Any file may be attached to the message, because mail does not look at the contents of the file. The attachment's main use is to send documents more complicated than plain text or a picture. The attachment was created with some document creation tool, and the same document creation tool will have to be used by the receiver to view the document. Since any file may be sent, the attachment may also be used for file transfer of software. The last use of attachments is to easily backup documents. The file to be backed up is attached to a message and sent to yourself. This file will remain in your mailbox on the mail server for future retrieval.
Implementation

Data for a message is stored in a mail object record of class "message". The envelope information is part of the record, because every message must have an envelope. The record also has "handles" for a text comment, a picture, and an attachment. A handle is a Macintosh term for a pointer to a pointer. Using handles allows the Macintosh memory manager to move a heap object without invalidating pointers to the object. A window is also associated with the mail message. One of the fields in the window is a handle to the message and one of the fields in the message points to the window. This simplifies sending events in a window to the proper object and reflecting changes to an object in the proper window. The window's rectangle is also kept in the message so that the message appears in the same place and as the same size when reopened.

The text comment is implemented using the Text Edit Manager. This part of the built-in Macintosh software handles simple editing and display of plain text.

The picture handle points to a Macintosh picture. The internal format of the picture is proprietary to Apple, yet they provide routines for creating and displaying these pictures.

The attachment handle points to a record containing the file type, file creator, filename, and then the file itself. The type, creator, and filename are needed by the receiver to properly save the attachment. Because the file itself is read into memory, this implementation will only work on files less than the size of free memory.

Results and comments

This message structure has been implemented and is flexible enough to handle the needs of most mail messages. Using text and pictures, messages can be sent which may be displayed without leaving the mail program.

This implementation is a weak version of insets. I already have a driver which displays TEXT and PICT data. In a future version, I would make it easy to add drivers of different types. These new drivers could be stored locally or on the mail server host. If a driver for an attachment existed, the attachment would be displayed. If a driver was not found, only the the attachment name and type would be displayed, as is done currently.

There are several uses for having more than one attachment. Sending a folder of files, or three related documents of different type are examples. I would agree to allow a folder or group of files if it did not become a complicated editing task to change the attachment list.

In my original design, messages had a hierarchical document structure. I no longer feel that this is the correct approach for two reasons. First, the mail program was becoming a complex editor of document pieces. In fact, the largest part of the program would have been the part that glued these different pieces into a message. I had
specifically stated that creating a complex editor was not one of my goals. Secondly, this tree editor would only be able to display plain text and pictures. The combination of these two data types did not seem to include most documents created on the Macintosh. Specifically, most documents are heavily formatted papers whose formatting is lost in conversion to plain text. People would not switch from using their word processor to using my mail editor to piece together their large papers.
The User Interface

Adding a Macintosh user interface is the second major goal of this thesis. A Macintosh user interface is quite different from Unix mail as is described in the background for this section. To more completely understand the design, it is useful to be familiar with the Macintosh and specifically the Macintosh User Interface Guidelines described in this background. Second, the actual design of the user interface is described. The objects displayed are explained as are the valid commands on those objects. Next, an explanation of the implementation is given. This section ends with a list of many alternatives tried and the reasons for not accepting them.

User Interface Background

Macintosh User Interface Guidelines

One of the main ideas that came out of developing the Macintosh is that it should have a consistent user interface. The Macintosh User Interface Guidelines [Apple] define what the consistent user interface should be.

The three overall qualities of the interface are responsiveness, permissiveness, and consistancy. A program is responsive if actions by the user cause something to happen. Inverting an item when it is selected is an expression of this idea. A permissive program lets the user do anything reasonable in order to make the user feel like he is in control. One way to allow the user to do anything reasonable is to eliminate as many modes as possible from the program. A consistent application behaves in the same way as every other application, so that the user does not need to remember separate interfaces.

These guidelines also give exact details on how to implement the selecting of objects, and how to use menus, icons, windows, dialogs, and controls.

Star Mail

The user interface to Star Mail [Xerox] is integrated into the desktop. Files can be sent by dragging them into the out box. All files must have a "cover sheet" to be mailed. This cover sheet contains addressing information. A cover sheet may also contain a short text comment and be sent by itself. Because mail is like any other file, it can be opened and read like any other Star file.

MacMail

This Macintosh mail system [Davis and Hourvitz] is very similar to the Finder. The Finder is the Macintosh program manipulating disk files [Kaehler]. This system was to connect to everything, and so it includes the ability to use features of many different mail servers. Some features address message delivery speed and format, and other features express whether a phone link, local network, or other connection is used to access the mail server. Addressing exhibits the same ideas as it tries to allow an
User Interface Design

Figure 4-1: The Startup Dialog

Running Mail

Mail appears to the user as a Macintosh Application. When the application is opened, a startup dialog is displayed showing the last connection. The user must fill in the entries to tell the program which mailbox he wishes to read. Though this project only attempted a server interface for Unix Mail, and the communications interface for a direct serial line, it is hoped the design of this program will make it easy to implement interfaces for all of the choices shown.
Figure 4-2: The Mail Desktop with the mss folder selected

**The DeskTop**

The mail desktop is extremely similar to the text view of a program called the Finder [Kaehler]. The Finder is the program which manages disk files, folders, and diskettes on the Macintosh. Diskette objects are replaced with a Mailbox object, folders are still folders, and file objects are replaced by message objects. There is a special kinds of folder - the in box. The in box is where any incoming mail is put. Like files in the "by name" view of the Finder, messages are seen with a small icon and a string of text attributes.

**Mail Icons**

As file icons show the type of the file, a message icon shows some field of the message. By default, the icon shows only the class of object: either folder or message. A field may be chosen and different icons for each value of that field may be made. For example, there may be different icons for opened and unopened messages, or different icons for messages from every different person who sends you mail.
Figure 4-3: The File Menu

The File Menu

The "new" command creates a blank message. This message can then be filled in and sent. An important idea is that there is no mode between received, sent, and unsent messages. The "open" command opens the selected object and the "close" command closes the selected object. Double clicking on folder icons is a shortcut for toggling the folder from open to closed or closed to open.

Messages (or any part of a message) may be explicitly saved as a local Macintosh file with the menu command "Save Local". This command shows a dialog prompting for a filename, and radio buttons for choosing how the file should be saved. The default filename is whatever the attachment had when it was sent. The "Text Only" radio button saves only the text comment of the file and any file that reads text files can open it. The "Attachment only" radio button saves the attachment using its own name, type, and creator.

"Add Attachment..." brings up the standard Macintosh "Get File" dialog. This dialog shows a list of all the files on the diskette. The user chooses a file by clicking on a filename and then clicking the attach button. This command attaches a file to the current message and the message display is updated to show the filename, creator and type.

"Page Setup..." and "Print..." are used to print out the current message on a printer.

"Quit" logs off the mail server and exits to the finder. If any messages have been created but not sent or created, a dialog appears for each one with buttons for save local, send, throw away, and cancel. These dialogs are needed because all new messages are would disappear unless explicitly saved or sent.
The "Edit" menu is used for all editing of messages. To perform an editing operation, a selection within the message is first made, and then an edit menu command (or its keyboard equivalent) is chosen. "Cut" removes the selection and puts it on the clipboard. "Copy" places a copy of the selection on the clipboard. "Paste" removes the current selection and replaces it with the contents of the clipboard. "Clear" removes the selection. "Select All" selects the largest current range but does nothing to the selection. Selections for items within the envelope and the text comment can be of any character range. For the picture and attachment, clicking anywhere within the item selects the entire item by outlining. Use of the menu bar by desk accessories is supported.

The View Menu

Figure 4-5: The view menu with all fields visible except "To".
All messages have envelopes which contain address and other fields. The view menu contains a checklist containing the union of these fields in all messages. A checkmark next to a field makes the field visible and no checkmark means the field is skipped and not displayed. Selecting a field toggles its value, so in Figure 4-5, the date field is about to become invisible. The default view for the mailbox is for all fields to be visible. The view within a folder inherits the default view from the mailbox, and the view within a message inherits the default view from its folder.

**Special**

<table>
<thead>
<tr>
<th>Change Mailbox...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reply</td>
</tr>
<tr>
<td>Read Next Message</td>
</tr>
</tbody>
</table>

Figure 4-6: The Special Menu

**The Special Menu**

The special menu offers shortcuts to functions which can be done in other ways. "Change Mailbox" is a command which is the same as quitting and starting again. This can be used to quickly change mailboxes. "Reply" takes the selected or open message and opens a copy of that message. The copy is identical, except that the "to" and "from" field values are switched and "re:" is prefixed to the value of the "what" field. "Read next message" closes the current message, and selects and opens the next message in the inbox. This command has been added, because sequentially reading new mail is such a common task.

**Enabling and Disabling Menu Items**

All menu items are not always available for use (enabled). When a menu item is unavailable, it is shaded grey, and will not become inverted upon clicking. As the frontmost window and the selected object are the object of almost all commands, they determine which menu items are enabled. A frontmost desk accessory causes only the edit menu and the close and quit items in the file menu to be enabled.

**Addressing**

The design of anything more than a typed address is incomplete. Part of this scheme depends on exactly how the global address server (in progress) is implemented. This directory will associate a user's name with the electronic mail address to which he would like to receive mail. Any non-ambiguous subset of the name will be converted into the proper mail address. I envision three ways in which addressing will change. The first way it will change is that mail will immediately notify the user if the address is invalid. This notification will either say that no address match was found or will allow the user to choose from a list of possibilities in the case of an ambiguous address. The second kind of addressing will be what most people think of as a mailing list. A copy of
the message will be sent to everyone on the selected mailing list. The third kind of addressing chooses only some names from a mailing list. The user will be shown the list and he will select which people will receive the message.

The user interface for the above mentioned addressing will activated by clicking a mailing list icon next to the "To" field. Clicking this icon will bring up a list showing all the current mailing lists. A mailing list could be selected, or a list could be opened and only a few people from the list could be selected. When a mailing list is opened, the display will be changed to show that the user is looking only within this list and not at all lists. This display will behave in the same way as the get file dialog for the new hierarchical file system. Addresses which are really names of mailing lists will a special icon to show that it is not just a person's name.

**User Interface Implementation**

The main Pascal program has an event loop which receives events from the Macintosh event manager. These events either select an object, or perform a command on an object. If an object is selected, the previous selection is unhighlighted, and the new selection is highlighted. If a command is received, the current selection is passed to the procedure which performs the command. This procedure checks the object class and in many cases passes this object to a more specific procedure. This idea of passing objects to messages is exactly the inverse of Smalltalk where messages are passed to objects. The reverse in implementation is caused by the Pascal language, but I can accomplish the same goals of hiding details and using code inherited among classes.

There are mail objects of class mailbox, folder, and message. These objects are ordered in a tree with the mailbox as the root, folders as its children, and messages as children of folders. The children of a folder are linked into a list with only the head of the list included in the parent. Each object also knows its parent owner of the window containing the object. Operations common to all objects are New, Open, Close, Drag, Resize, Draw, Invalidate, Invert, Fill, Add, Delete, and Move. These operations are self-explanatory except for invalidate and fill. Invalidate means that the object has changed and should be redrawn. Fill means to get information about this object from the mail server. Using a common routine for these operations enabled me to simplify code with recursion. For example, the procedure to close an object is to recursively close all its children and then close the object itself. This same close procedure works for mailboxes, folders, or messages.

Open messages can have the additional editing operations send, fill, cut, copy, and paste. The current selection within the message is then used to determine exactly what is changed.

Editing within the message is different depending on in which part of the message changes are being made. The program uses rectangles which differentiate between the envelope, the text comment, the attachment, and the picture. The dialog manager is used to handle events in the envelope. Dynamic envelopes are not implemented,
but can be implemented by changing the dialog item list in memory. This is what REdit and ResEdit do to manipulate dialogs. The mailing list function for the To function is also not implemented. Hopefully, the standard get file routine could be used by fooling it into thinking that the mailing list is a list of files.

If dynamic envelopes are available, then the view menu could be completely implemented. Inheriting views is possible by searching up the tree until a non-default entry is found. The view menu list is created empty initially, and is updated every time the contents of a new envelope are received.

**User Interface Results and Comments**

A user interface has been developed which follows the Macintosh User Interface Guidelines. This interface is simple, and is quite powerful. It is similar enough to the Finder interface and interface for other programs that users can quickly learn to use mail.

In creating the user interface, many ideas were tried. Some ideas were successful and some were not. I believe this experimentation is essential until building a user interface has been more thoroughly developed into a science.

One implementation which failed was giving a window to every open folder. This is how the Finder operates and it seemed that the analogy was similar. The result was that there were too many windows on the screen. Trying to fit every window with its own border, scroll bars, and resize box on the small Macintosh screen didn't work. A similar problem came up with having multiple mailboxes. My first implementation had a desktop window which could contain many mailbox icons. This concept added an extra level of hierarchy (and a window) to the mail program. I replaced the desktop window with a "Change Mailbox..." command. Though this command makes changing between mailboxes more difficult, the need for multiple mailboxes should diminish with the addition of the global name directory.

In mimicking the finder again, I came up with another discarded idea. I originally included both a textual and iconic view of mail. The iconic view showed the object icon and its name arbitrarily ordered on the screen. The text view is the view in the current design. The problem with the iconic view is that just the name was not enough to describe the message. I also wanted to know the values of other fields. I also do not agree that the flexibility of moving icons to any place in the window is needed. I found that I preferred the current version which always automatically organized my messages.

There were several variations of an out box possible, so because of the confusion, I decided to not have an out box at all. One option for an out box is to have any mail placed within it immediately sent. Another idea is to place messages into the out box and then explicitly send all messages with an "Empty Outbox" command. A third idea is that the out box is just a log of all mail sent. I felt that there were good and bad reasons for all of these ideas, so I chose to use the unambiguous "Send" command in
place of any out box whatsoever. Any functionality lost by having only the "Send" command can be provided by a Unix mail setting which puts a copy of all sent mail into a specified folder (e.g. "Sent Mail Log").

Automatically moving messages that have been read from an "unread" folder into a "read but unfiled" folder is another idea I rejected. I chose not to incorporate this idea because it is not obvious to the user what is happening to the message. However, I do think that a difference in icon between read and unread messages could be useful.

Mail and the Finder are two separate modes. The finder is such a complicated and undocumented beast that it would be extremely difficult to integrate the two programs without the source for the Finder. I still prefer an integrated solution, but have conceded that it is not a practical goal.

This application was not written as a desk accessory because of the added complexity desk accessories involve. There is no fundamental reason why this could not be made into as a desk accessory, except that caution must be used not to use too much memory. This program does work very nicely with the Switcher and provides almost the same utility as a desk accessory.
The Communications Interface

There are many reasons why the Macintosh must communicate with a mail server instead of mailing directly to the recipient. Most importantly, users do not always use the same Macintosh. It would be unreasonable to require students to use the same Macintosh in a public terminal cluster. Multi-tasking and constant operation are two possible but impractical requirements for mail delivery. Receiving messages also requires storage. Using RAM is unacceptable because, it is volatile and needed by other applications. The floppy disk is not acceptable storage because it may be removed. A hard disk would work but only a small percentage of users have hard disks.

Using a server also allows further flexibility. Mail can be sent to a known location, and then the user can request mail as needed. Macintosches in different locations and even other terminals or workstations could all be used to read a user's mail file.

With interfaces of any type, layering is a good method. Layering's advantages are the forced separation between the two sections, and the hiding of details. Examples of layering are seen throughout communications standards and structured programming.

Layering the Communications Interface increases adaptability - one of the minor thesis goals. Providing a clean interface strictly defines the only part of the program which needs to be changed.

Background on different communications interfaces is described followed by a description of the two different interfaces needed to maximise adaptability. The first interface is between procedures handling mail objects and procedures sending and receiving these mail objects. This layer hides details of logging onto the mail server, sending messages, and receiving messages. The second interface is between the real communications link and an invented stream file. This layer hides details of sending and receiving characters, bytes, words, and lines. The design, implementation, and results are discussed for each interface.

Background

Remote Procedure Calls

Researchers at Xerox developed the idea of Remote Procedure Calls (RPCs) in order to simplify the writing of Ethernet-based software. The idea is to treat a request over the network in the same way as a local procedure call. In the first step of a remote procedure call, the call and parameters are passed to the local RPC handler. This handler knows the types expected for each call, converts these types into a Xerox RPC standard, and stuffs them into a packet. The Packet is sent to the server containing the call and the converted parameters. Upon delivery, the server RPC handler converts the parameters into the appropriate local format and passes the parameters to the appropriate routine. The called procedure runs, completes, and returns any results to the server RPC handler. The server RPC handler converts the results into the standard
format, and sends a return packet. The local RPC handler receives this packet, and
decodes and returns the result to the original calling procedure. An additional very
important feature was that at link time, type checking of parameters was done over the
network in order to validate RPCs.

User Interface is a Separate Entity

Many articles choose to split the program into a user interface section and other
sections. This piece should be loosely connected and communicates only thorough its
interface. MacMail explains the idea as a user agent communicating with its mail
angel. One of the goals in a mail system at CMU [Borenstein and Rosenberg] was to
have common lower lever routines on top of which several different interfaces could be
built.

Mail Server Interface Design

The Mail server interface is used by the routines handling mail objects. When an
operation is to be performed on a Mail Object, a check is made to insure that all the
necessary attributes of the mail object exist. If they do not exist, the object is passed to
a mail server procedure capable of retrieving the needed information from the mail
server. For example, opening a new mailbox, triggers the retrieval of the folders within
that mailbox, and opening a message for the first time, causes the contents of that
message to be retrieved. Mail object information is retained in local memory in order to
minimize retrieval time. In case of a memory shortage, the mail object information
could be purged, because it would be retrieved again whenever it was needed. The
entire mailbox file could be read at startup, but efforts to minimize memory needs and
startup time force retrieval of only the smallest amounts of information at any one time.

The mail server interface was designed so as to not depend heavily on any special
features on Unix mail. Unix mail was the only mail system for which the mail server
interface was written. The self-contained mail server interface should allow almost any	imesharing mail system (including DCTS) to be used in its place. Unix mail was
chosen because of its widespread use, especially in the academic community. It was
hoped that choosing a common mail server would make this project of more interest to
those outside of the Dartmouth community.

Mail Server Interface Implementation

The most interesting part of the Mail server interface is how a complex message is
received. First, the text comment is received verbatim from the server until a line is
found signaling the beginning of a picture, the beginning of an attachment, or the end
of the message. After the start character of a picture or an attachment, there is the
length of the item. For attachments, three strings - filename, filetype, and creator - are
listed next. Then, for both pictures and attachments, the length of the data is received,
and that many bytes of data is put into the respective handle.

Mail Server Interface Comments
For text comments, the data is sent and received a line at a time. For pictures and files, the data is sent a byte at a time. The speed of sending would be greatly increased if pictures, files, and text were sent in much larger chunks.

Stream File Design

This layer's most important job is to convert arbitrary bytes into characters. This conversion is necessary because UNIX messages may contain only a subset of the full 256\(^{10}\) possible byte combinations. Codes over 128\(^{10}\) are not allowed because the high bit is sometimes used for parity and sometimes ignored. Some codes under 32\(^{10}\) may not be used because they implement flow control or other network features and 127\(^{10}\) may not be used because it is the rubout character. Even if the characters did correctly cross the network, MAIL is not written to handle binary file transfer.

A simple escape scheme is used to code the bytes. Bytes in the acceptable range are sent as their character equivalent. Unacceptable bytes are encoded using predefined escape characters followed by the byte +/- a predefined offset. This crude but simple algorithm was chosen because a specific non-goal of this thesis was the optimal coding of files. Lines containing bytes converted into character are truncated with a carriage return after a predefined length. Truncating lines avoids problems with maximum line lengths. When receiving bytes, this carriage return is ignored.

In addition to bytes, words, quoted strings, lines, and commands can be sent and received. These items are all self-explanatory except for commands. Sending a command, sends the first string, and then waits until the second string (usually a prompt) is received. "Peeking", also part of the stream file interface, at the character about to be read is a great aid to the parsing task of the mail server interface.

The first communications interface implemented is an RS-232 serial link, though it is hoped that it will not be difficult to implement a stream file interface for a different communications link. This communications standard was chosen because it is the lowest common denominator. As a similar stream protocol (Kiewit Stream Protocol) already exists for Appletalk, it should be a trivial task to convert this program to run on Dartmouth's Appletalk network (Kiewit network).

Stream File Implementation Comments

I have used the idea of a stream file twice before in Macintosh front-ends to other computers. The first time I developed this idea was in an attempt at a graphical interface to a Unix version of Smalltalk. The second use of the stream file concept was a graphical interface to a minicomputer running a digital "tape" editor. The Smalltalk interface also uses the simple encoding method mentioned to transfer bitmaps. The only large change was the addition of the Peek character. Small changes were made to make the stream file interface into a Lisa Pascal UNIT, and other changes were made to improve generality.
Using the Lisa as a mimic mail server turned out to be extremely useful. A serial cable between the Macintosh and the Lisa was connected. The Macintosh mail program would be run and then the Lisa, using a terminal emulator, would imitate a Unix system. Use of script files on the Lisa automated simple parts, but I still retained complete control from the keyboard. In a similar fashion, I modified the Macintosh read and write routines to also use characters from the keyboard. I have complete control on both sides of the communications interface.

**Stream File Results**

Many layers of nested procedures and static strings make this section extremely general, but also inefficient. Strings are such a cautious item because they are statically declared and use 256 bytes even if they are empty. Multiple concatenations cause unnecessary copying and many levels of procedure stack frames waste memory. Even with these faults, this UNIT still seems to keep up with the server, but it would be the first one in which I would attempt optimizations.
Overall Results

A powerful yet simple prototype has been implemented that satisfies the two major goals - a Macintosh user interface to mail and complex documents. This prototype can currently log onto the Unix host, startup Unix mail, and retrieve messages and folders. Messages with a text comment, a picture, and an arbitrary attachment can be read, created, edited or sent. Editing supports full use of the clipboard and cut, copy, and paste.

Several parts were not completely implemented. Most of the work left is concentrated in adding capabilities to the Envelope. Adding new fields of arbitrary name, and adjusting the View menu to handle these fields still needs to be done. Addressing more complicated than a typed string was not implemented. Conversing with the global mail directory, sending to distribution lists, and creating distribution lists still needs to be done. Vertical and horizontal scroll bars were not implemented, but their effect on the prototype was minor.

The major shortcoming of this project is that there was not time to add browsers of arbitrary type. These browsers would know how to display (and possibly edit?) attachments of a certain type. If the browser for an attachment existed, the attachment would be displayed. Otherwise, the attachment filename, creator, and type would be shown along with a message saying that it could not be displayed. These browsers would most likely be stored on local storage, but could also be stored on the mail server.

Until this project had been implemented, it was not known if a Macintosh interface could be of acceptable speed. Experiments with this project seem to indicate that the speed of the program is largely limited by the speed of communications. This promises to make a Macintosh interface at least as fast as its terminal counterpart. Files of text, outline drawings, and other densely coded files are quickly sent and received. Bitmaps and applications are probably still too large to send often, but they could be sent on rare occasions.

This mail system has an interface consistent with Macintosh guidelines, supports transfer of multimedia Macintosh files, and is of acceptable speed. These features will allow it to replace both terminal emulators and plain terminals as the program of choice for mail.

Future Work

The many opportunities for additional work on mail systems will be strongly influenced by efforts to tightly bind workstations with other computing services.

An accepted document standard is strongly needed before disparate machines can be tightly interconnected. This document standard will define complex documents containing different text fonts, text styles, pictures, voice, and other digital information. When this document standard is accepted, mail systems will be changed to understand
how to display and possibly edit these documents. Apple has already made this step on a smaller scale by defining their text and picture formats. These formats make it easier for separate applications to work together and exchange information. Much more powerful standards need to be accepted to encourage exchange of many more types of digital documents.

As workstations become more closely connected, the differentiation between your own resources and someone else's will diminish. An external file system will make the difference between your local files and those on a host disappear. Specifically, the external file system would replace much of the server interface and network communications code in this mail project. Mail would be changed into a program which manipulated files and directories. The distinction between local and remote processes will also disappear. It may soon be possible to run many programs on many different hosts using a consistent Macintosh interface. This specific mail system is a front end to an existing timesharing mail system. Though it would be possible to construct distinct interfaces for all timesharing programs, a generalized timesharing interface can be built which simplifies the process. Apple is attempting to build such a program called MacWorkstation [Apple]. Apple's program receives commands from the host to draw windows, menus, and icons from the host, and returns mouseclicks, and other events to the host.

Just as distinctions between local and remote resources disappear, differences between documents, mail, and other files will also disappear and all these items will be subject to powerful uniform manipulation. In the Xerox Star office system, a mail file is any desktop file with the addition of some addressing information. A uniform design reduces modality as the user never has to "enter" mail to send or receive a message. One way that these documents could be manipulated is if they were all entries in a database. Both HERMES, a message system, and PROFFS, a document system, allow simple sorting, searching, and categorizing. Other powerful manipulations, taken from NLS, such as tree structures and links between files might also be added.
Bibliography

Annotated Bibliography

Documents
User Interface
Communications Interface
Documents


Coutaz, J., "A Layout Abstraction For User-System Interface", *ACM SIGCHI bulletin*, v.16 n.3, January 1985, pp. 18-24. Coutaz defines a box as something containing data. Many views of this data are possible. The user can change the views without changing the data. Both the data procedure, and the view procedure are something given to the system, and then updating the window is performed automatically.

Crocker, David H., "Standard for the format of ARPA internet text messages", RFC 822, University of Delaware. A new standard is proposed for messages sent over the ARPA internet. Header fields are defined, but the structure of the message, assumed to be text, is not defined.

Davis, Mark, and Hourvitz, Leo, "Macintosh Mail Specification", Apple Computer, Cupertino, California, 1985. This is the complete functional specification and design for the planned Macintosh Office mail system. This project was cancelled due to problems with the file server. The system was to use a file server (connected over Appletalk) to hold and receive messages and as a gateway to other networks. The user interface is similar to the finder with folders, documents, and some flexibility of views. Messages have headers with fields, a short text message, and attachments of arbitrary type. Eventually, messages in this system would conform to the x.400 message standard. New messages are chosen from a list of available templates (e.g. invoice, phone message, or memo). Options are included to work with all the variations of commercial services (MCI mail speed, type, logo, and signature). Sending is done by selecting messages and then choosing a send command. No info on sent messages is kept except a message ID number. Recipients may be typed or chosen from an address book. The address book contains all addresses (private, local server, and other servers). Addresses can be grouped into Distribution lists which may contain other distribution lists. Many different mail boxes can be configured, and these mailboxes can even be on non-Apple servers such as MCI mail.

Deutsch, Debra, "Design of a message format standard", *Computer Message Systems*, North-Holland, N.Y. 1981, pp. 199-220. The future NBS standard for messages. A document is a series of fields; there is no distinction between the header and contents. Each field contains its length so that unknown fields can be skipped over. Documents are not layered. The standard uses a machine readable format. They discuss the difference between syntactic and semantic reissuing. Semantic reissuing may be ambiguous if (like in the standard) the order of fields is not guaranteed. (e.g. it is not known who sent you a message with multiple "reissued
to:"

Feiner, Steven, Nagy, Sandor, and van Dam, Andries, "An Integrated System for Creating and Presenting Complex-Based Documents", Computer Graphics, v15 n3, Siggraph 81 Conference Proceedings, 1981. The paper describes a NLS style document including high resolution color pictures. Links between documents and chapters may be jumped by touching a hot spot. They plan on using a similar system for expert systems and explanation manuals.

Goldberg, Adele, and Robson, David, Smalltalk - 80: The Language and its Implementation, Addison-Wesley, Reading, MA, 1983. This book describes the Smalltalk-80 language and how to implement its interpreter. Smalltalk is best known as an object-oriented language. Actions are performed by sending messages to objects. The object looks to its class to see if it can understand the message, and if not passes the message to its superclass. It is easy to hide data and create general reusable objects. Smalltalk is also a complete environment and many ideas about windows, bitmapped-graphics, icons, and menus came from research done with Smalltalk.

Hansen, Fred, "Andrew insets", personal communciation, winter 1986. The Andrew workstation will have drivers for different insets. A window is a superset of an inset. Insets may be nested. A driver will completely handle the inset. For example, a PICT driver may not only know how to draw a PICT, but may also let the user edit it. The importance of drivers is that this driver will be used anytime a PICT inset receives an event. Drivers may be system drivers, group drivers, or personal drivers. Currently, there is no protection from a malicious driver.

Horak, W., and Kronert, G., "An Object-Oriented Document Architecture Model for Processing and Interchange of Documents", ACM SIGOA v.5 Nos. 1-2, (June 1984). The future CCITT (also X.400) standard for both revisable form and final form documents is explained. Documents are trees in both forms. Logical objects are paragraphs, chapters, sentences, etc... Layout objects are page margins, line spacing, etc... The logical objects give the data, and the layout objects describes the view of the data.

Newman, Julian, "Human Factors Requirements for Managerial Use of Computer Message Systems", Computer Message Systems, North-Holland, N.Y., 1981, pp. 453-465. Managers won't use computers unless they become better at getting information, and manipulating this information. They need to be able to change details like structure, spacing, and levels of detail. Documents need to have layout values which can be changed without changing information content.

Parks, Raymond R., "Electronic Mail", Datamation, October 1, 1984, pp. 118-122. Parks compares IBM DIA/DCA to X.400. IBM standard does not use header fields, so must look at document contents to search attributes. X.400 has many header fields and then does not look at contents. Both these standards (and other
existing standards) ignore the format of distribution lists.

Postel, J., "A Structured format for transmission of multimedia mail documents", RFC 767, USC/Information Sciences Institute, August 1980. A very general structure for multimedia mail documents is described. Any media type is possible, and the media are ordered in lists. Each list is a new object which can be an object in another list. These lists can be declared to be of type sequential, simultaneous, or unordered. Infinite levels are possible and NLS files can be easily converted to this format. The individual formats for the media are suggested but not declared. Instead, this format gives a structure for piecing together different media.

Richardson, Bronmyn, "Documentation for Macwrite file format: version 3.0 or greater", Cupertino, California: Encore Systems, December 4, 1984. The exact format of a MacWrite file is given.

Routhorn, G.A., and Carruthers, P.A., "Teletex and its protocols", Computer Message Systems, North-Holland, N.Y., 1981, pp. 259-283. This explains the teletex protocols. Documents have some structure (pages). The negotiation of terminal capabilities is also interesting in that a similar process could be used with more complicated documents.

Tapscott, Don, "Research on the impact of office information communication systems", Computer Message Systems, North-Holland, N.Y., 1981, pp. 395-409. This is a study of change in work by after given a message system. Results show that more work was done, less time wasted. Messages partially replaced one-to-one meetings and phone messages.

Think Technologies, Thinktank reference manual, Macintosh version. Thinktank is an outlining program. Many levels of structure are allowed. Clicking on a level alternates between opening and closing that level. The program can move and copy parts of the tree from one place to another. Outlines are limited to plain text.

Vittal, John, Active Message Processing: Messages as Messengers, Computer Message Systems, North-Holland, N.Y., 1981, pp. 175-195. Messages need not be passive, but may contain instructions that are performed when they are "read." In Vittal's active messages, instructions for composing new messages, querying the user, and selecting messages are part of the message. These instructions may also be used to change the user environment by creating new commands, or automatically filing new messages into certain folders. Examples are giving showing how active messages can be used to produce a questionnaire, a calendar scheduler, and a complicated purchase order.

Wilder, H.A., and Maxemchuk, N.F., "Virtual Editing: II. The User Interface", ACM SIGOA Newsletter, v.3 n.1and2, (June 1982). pp. 41-46. They explain the interface for editing a document composed of a list of text and voice pieces. Separate drivers for each media exist, but they use common functions. There is a time line for voice. Icons show relative sizes of texts. This interface allowed editing of the list
of text and voice pieces, but one cannot edit within the pieces.

Zisman, Michael, "Good fits, bad fits, and misfits", ComputerWorld, January 21, 1985, pp. ID 1-12. Revisable and fixed form documents are described. It is very difficult to fully translate documents and also keep both visual and logical meanings. Translating from source to canonical form, and then translating to destination form reduces number of translators needed from $n^2$ to $2n$. 
User Interface

Apple Computer, "AppleLink" (software), Cupertino, California, 1985. This software provides a Macintosh interface to a variety of mainframe services, databases, and mail. Icons, menus, and TextEdit editing is used to replace a command driven interface. Because connection time to the mainframe is slow and costs money, an effort is made to do work offline.

Apple Computer, "Macintosh User Interface Guidelines", Inside Macintosh, Cupertino, California, 1984. This chapter describes the parts of the Macintosh user interfaces that should be the same throughout all programs. This document is to be the final judge of how a certain feature should be implemented. Other general guidelines for good user interfaces are also suggested.

Apple Computer, "MacWorkstation" (software), MIS department, 1986. This product adds a Macintosh interface to a UNIX or VMS application. The connection is a serial line. Menus, windows, dialogs, and Macintosh quickdraw routines can be displayed, and events such as menu choices and completed dialogs are returned. The MacWorkstation software is dumb and knows nothing about the remote application which is being run.

Borenstein, Nathaniel S., "The Bags Message Management System User's Manual", Carnegie-Mellon University Computer Science Department, 1985. Bags is a message system which runs under Emacs, a very extensible screen editor. Like Emacs, Bags can be customized and provides hooks to do so. This manual describes those hooks as well as all its normal and obscure features. The most impressive feature is that it handles mail, bulletin boards, and news in a common interface.

Borenstein, Nathaniel S, and Rosenberg, Jonathon, "Electronic Communications: A Wish List", Information Technology Center, Carnegie Mellon, data unknown. Before building a new mail system at Carnegie Mellon, the authors attempt to define an ideal system. Their complete system would include mail, conferencing, bulletin boards, calendars, distribution lists, and news groups in one unified structure. Messages will be classified by several keywords. Messages are read by giving a search similar to a database command. Reviews of messages will be made and then the review becomes another attribute that can be searched on. The user interface should be completely independent of the other mail software.

Bruder, J., Moy, M., Mueller, A., and Danielson, R., "User experience from the design of a local electronic mail system", Computer Message Systems, North-Holland, N.Y., 1981, pp. 69-78. Study showed that most messages were short and users used read, send and delete commands a lot, but didn't use other commands. Future improvements will be to make mail one unified mail system and to add a built-in text editor.
Card, Stuart K., Moran, Thomas P. and Newell, Allen, *The Psychology of Human-Computer Interaction*, Lawrence Erlbaum Associates, Hillsdale, New Jersey, 1983. Using models of users, tasks, and methods to do these tasks, they perform an in-depth study of user interfaces to text editing. They tried to scientifically study these problems like one would do psychological research. They had advice for system designers: 1) Early in the system design process, consider the psychology of the user and design of the user interface. 2) Specify the performance requirements, the tasks, and the methods 3) Match the method anlaysis to the level of commitment in the design process. 4) Eliminate operators from the method of doing tasks. 5) Design error recovery methods

Cohen, Danny, "A Voice Message System", *Computer Message Systems*, North-Holland, N.Y., 1981, pp. 17-28. Over ARPANET, Cohen has developed a reliable voice message system. Because voice messages are so large, the message may have to be stored at a third system temporarily, until the recipient has enough file space. He also has created a voice interface to text mail, and a text interface to voice mail. These simple interfaces allow someone who only has access to a voice terminal to check if he has text mail and someone who only has a text terminal check to see if he has voice mail.


Daney, Charles, "The VMSHARE computer conference facility", *Computer Message Systems*, North-Holland, N.Y., 1981, pp. 115-128. This is a message system on a single mainframe. Messages are files in directories. Anyone can write to directories, but only the user may read his directory. The system needs an index of messages with complex search capabilities.

Davis, Mark, and Hourvitz, Leo, "Macintosh Mail Specification", Apple Computer, Cupertino, California, 1985. This is the complete functional specification and design for the planned Macintosh Office mail system. This project was cancelled due to problems with the file server. The system was to use a file server (connected over Appletalk) to hold and receive messages and as a gateway to other networks. The user interface is similar to the finder with folders, documents, and some flexibility of views. Messages have headers with fields, a short text message, and attachments of arbitrary type. Eventually, messages in this system would conform to the x.400 message standard. New messages are chosen from a list of available templates (e.g. invoice, phone message, or memo) Options are included to work with all the variations of commercial services (MCI mail speed, type, logo, and signature). Sending is done by selecting messages and then choosing a send command. No info on sent messages is kept except a message ID number. Recipients may be typed or chosen from an address book. The address book contains all addresses (private, local server, and other servers). Addresses can be grouped in Distribution lists which may contain other distribution lists. Many different mail boxes can be configured, and these
mailboxes can even be on non-Apple servers such as MCI mail.

Dawes, N.W., Harris, S.J., Magoon, M.I., Maveety, S.J., and Petty, D.J., "The design and service impact of COCOS, an electronic office system". *Computer Message Systems*, North-Holland, N.Y., 1981, pp. 372-384. This message system has addressing similar to the post office. Multiple variations of addresses are allowed. There is a global directory, and any match is a valid address. If possible, ambiguous addresses are shown to the user for further action, but otherwise the message is sent to the closest department paper mail office for personal delivery. System included macros, and a editor with a formatter. The poor use of computer conferencing made them conclude that all functions of conferencing could be done with messages.

Digital Equipment Corporation, *ALL-IN-A Office Menu User's Reference*, Volume 1-2, 1985. Part of DEC's office system includes a mail system with folders for unread, read, unsent, sent, and other mail. Messages have one possible attachment. The unique features are "defer delivery until date", "schedule meetings", and automatic reply.

Ellis, Clarence A., and Bernal, Marc, "Officetalk-D: An Experimental Office Informational System", *ACM SIGOA Newsletter*, v.3 n.1 and 2, (June 1982), pp. 131-137. This is the second version of Officetalk-zero. Global aids were added which enabled managers to keep track of workers. Alerts could be set up to go off on certain database or time values. The emphasis was on distributed actions and people working together. Also included a powerful "journaling" mechanism that translated actions into an English-like script language.

Good, Michael, "Etude and the Folklore of User Interface Design", *ACM SIGOA Newsletter*, v.2 n 1 and 2, (spring/summer 1981), pp.34-43. In discussing the design of his editor, Good does a survey of user interface research. Select responses, undo, cancel, help, and quick response time are all items important to a good user interface. Good (ha, ha) references are listed.

Good, Michael, Whiteside, John, Wixon, Dennis, and Jones, Sandra. "Building a User-Defined Interface", *CACM*, October 1984, pp. 1032-1043. Subjects were given a command driven mail system with no instructions and told to perform a set of tasks. After each iteration, the system was adapted to accept the commands that the subjects had tried. Results showed that by including command synonyms and increasing flexibility of command order, about 90% of user input was successfully parsed and executed. This result also applied to experienced users, though experienced users deviated less. They noticed that users often mimicked what was displayed, so statements displayed should be similar to commands. Conclusion: Iterative testing of user interface by users is important. Exception: Unfamiliar tasks like a debugger are quite different from a familiar task like mail. Good References are listed.

Haigh, Robert W., Gerbner, George, and Byrne, Richard B., *Communications in the 21st*
Century, NY, NY: John Wiley and Sons, 1981. The common prediction is that people will have an overwhelming choice of video and text media. Human-computer interfaces will improve to handle this overload of information. One will be able to (and will have to) choose exactly what topics one wants to know about.

Information System Operation, "CRDMail, A user's guide", General Electric Corporate research and development, Schenectady, NY. This manual describes how to use the center-wide CRDmail and the global name directory from VAX VMS and IBM VM. Text only messages are allowed.

IBM, Using the Professional Office System, VM/System Product, Publication Number SH20-5604-0, 1983. PROFFS is a system to keep track of documents. As a document is being created, different people have control and these privileges are "sent" between users. Documents are long, versions are kept, only one copy of the document exists, and only certain people may change the document. A log of incoming and outgoing documents is kept.

Kaehler, Carol, Macintosh, Cupertino, California: Apple Computer, 1984. The owner's manual for Macintosh which includes a chapter (3) describing the Finder, the Macintosh program which organizes files on the desktop.

Katz, A. "An experimental internetwork multimedia mail system", Proceedings of the IFIP 6.5 working conference on computer message services, May 1984. The author has developed a electronic mail system for messages containing both voice, bitmap, and text data. The system can create multimedia documents containing a list of voice, text, and bitmap items. Items can only be pieced together in the sequential fashion.

Kiewit Professional Staff, A Macintosh interface to Mail, Informal Meeting, Hanover, New Hampshire, January 22, 1986. Staff from user services and development were shown an early paper version of this project and were asked for comments. Their opinion was that text only messages were satisfactory. Additional functionality should be sacrificed for ease of use. The scheme for addressing presented was unsatisfactory.

McComb, Gordon, "Review: Electronic Mail - Special Delivery", MacWorld, December 1985, pp. 82-85. This is a review of Mail Center by Videx. System receives messages while other Mac programs are running. There is a send log, receive log, a window of on-line users and from, date, time, and status attributes. The Subject Attribute can be read by opening selected messages. Standard Macintosh dialog is used to choose file to send and From, To, Re, attributes are attached to the message. One must leave mail and startup a application to read or compose messages. A desk accessory notifies when mail is received. There are not any distribution lists. In order to quickly create/read short text only messages, a simple text editor should be added to the system, or the Switcher and an editor should be used.
Meads, Jon, "Report on the SIGCHI Workshop on Planning for User Interface Standards", *ACM SIGCHI*, v.17 n.2, (October 1982), pp. 11,16. Because so little is known about measuring the quality of a user interface, technology is advancing so rapidly, and there is so little experience with user interface systems, a standard would do more harm than good.

"mh", man files of The Rand MH Message Handling System, Sun Release 2.0, Sun Microsystems, Atlanta, Georgia. This Unix mail system is like "mail", but each function is a separate program instead of one large program. The data format of mh files is optimized better than the unformatted text file used by "mail". The many combinations of commands is very powerful, and new ones can be created because the file formats are given.

Morgan, Chris, "An interview with Wayne Rosling, Bruce Daniels, and Larry Tesler", *Byte Magazine*, February, 1983. This is an interview with the people in charge of the design of the Apple Lisa. First, the user interface was designed. Next, the software was designed, and then the hardware was designed. In developing the different user interface, many ideas were gathered, several prototypes were implemented, and then unskilled users tested the prototypes. This iteration happened many times. The idea of an iconic desktop was borrowed from the Xerox Star, but most other features were taken from Smalltalk. The Lisa is a level ahead of the Star, because it's primitive graphic operations know about curves, while the Star can only do rectangles. The development toolkit is a very important item, because it enables third party developers to leverage off research done by Apple.

Morris, Satyanarayanan, Conner, Howard, Rosenthal, and Smith, *Andrew: A distributed personal computing environment*, Information Technology Center, Carnegie-Mellon University, November 1985. Andrew will be a system of powerful workstations connected to large file servers and a communications network. It is designed to be portable (C and UNIX) among workstations. It is a combination of the shared device benefits of timesharing and the responsiveness of personal computing. An windowed/bitmapped interface for multiple processes is available.

Nafis, Christopher, Davidsen, William, Scavullo, Vince, and Meenan, Peter, "Electronic Mail Among Heterogeneous Computers", draft version, Information System Operation, Corporate Research and Development, General Electric, Schenectady, NY, 1986. To provide mail facilities for a collection of VMS, UNIX, and VM hosts, a global name directory was created with the person's name and the address at which he wanted to accept mail. Ambiguous names cause a list to be returned to the user from which he may choose. In addition, a portable user agent was written that had more capability than VMS mail. This enables a potentially common interface to mail as they already have a common editor that has been implemented for all machines.

Pearson, Michael M.L., and Kulp, James E., "Creating an adaptive computerized conferencing system on Unix", *Computer Message Systems*, North-Holland, N.Y., 1981, pp. 129-145. They rapidly implemented a single machine mail system using the c-shell, messages as files, and a hierarchical file system. Sending a message is the same as copying a file into a directory.

Postel, J. et al, "Internet Concepts Research", Annual Technical Report, USC/Information Sciences Institute, ISI/SR-85-150, 1984, pp. 53-65. This is a summary of work on an internetwork multimedia mail system. Voice, text, and bitmaps have been sent. Protocols for multimedia mail documents have been developed. Work is planned on a mail user agent using the xerox 8010 (star) and interlisp D.

Shoens, Kurt, "MAIL", version 2.18, Unix Programmer's Manual, 4.2 BSD Unix, /usr/doc/Mail. This manual describes the Unix "mail" system. Text files can be sent with arbitrary headers. Messages can be read, and also stored in folders. Messages can be sent through many networks of other Unix machines.

Scragg, Greg W., "Some thoughts on paper notes and electronic messages", *ACM SIGCHI*, v.16 n.3 (January 1985), pp. 41-44. Post-Its are compared to e-mail messages with the conclusion that Post-Its have many features that future e-mail systems should have. Messages should be highly visible, quick to produce, easy to organize, and easy to attach to other objects. As Post-Its can be put on coffee machines giving a warning or instructions, messages should also be attachable to objects (documents, programs) to give warnings or special instructions.

Simmons, Chuck, "Problems with DCTS mail", personal communication, 1/21/86. This is a list of the problems with the mail system on Dartmouth timesharing from the current author/revisor of the program. One problem is that the mailbox is in terminal format (text only). This makes it hard to store non-text information, and to randomly select messages. Creating and sending a message are linked as one operation. Mailboxes are part of a users catalog, so mail must verify the format of the mailbox and worry about the user changing permissions on the mailbox. One new suggested feature is an automatic reply similar to that of DEC's All-in-one system. This could be used to immediately send a reply to all received messages saying "I'm on vacation until March 3rd."

Smith, David Canfield, Irby, Charles, Kimball, Charles, Verplank, Bill, and Harlslem, Eric, "Designing the Star User Interface", *BYTE Magazine*, April 1982. Fundamental concepts of the user interface were designed before any hardware was built or software was written. Task analysis of what users want to do is
important. They also did a lot of prototyping, and they split concepts into easy and hard classes. When possible, they used a conceptual model familiar to the user. Seeing and pointing is better than remembering and typing. Other main concepts are WYSIWYG, universal commands, consistency, simplicity, modeless interaction, and user tailorable.


Turoff, Murray and Hiltz, Star Roxanne, The Network Nation: Human Communication via Computer, Reading, Mass.: Addison-Wesley, 1978. This paper describes how computer conferencing will change the country over the next twenty years. EIES is a service to make it easier for scientists to distribute information without the lag time needed by publications. It is used for messages, conferences, notebooks, bulletins. They see group communication as much more important in the future than simple message systems. They also describe a past system, EMISARI, successfully used by the government to quickly receive, organize, and distribute reports.


Uhlig, Ronald, Farber, David, and Bair, James, The Office of the Future, North Holland, NY, NY, 1979. This book looks at uses of computers in the office, the current technology, and the impact of office automation. Their main point is that many different tasks use a common set of tools for communicating, gathering, retrieving, analyzing, organizing, transforming, generating, modifying, and filing. Powerful tools should be developed for computers fulfilling these general needs instead of specific solutions.

Vittal, John, "MSG: A Simple Message System", Computer Message Systems, North-Holland, N.Y., 1981, pp. 329-343. This is a description of the user interface of one of the first message systems. He compares MSG to other early systems, and gives rough descriptions of those other systems.

Wegmann, Alain, "Vitrail: A Window Manager for an Office Information system", ACM SIGOA newsletter, v.5 n 1 and 2 (June 1984), pp. 1-12. Wegmann describes a multiple concurrent window, menu, and event manager for the Kayak workstation. User window procedures call Vitrail and are blocked until an event is received for them. The System includes standard structures for graphics and text and transfer of these structures between applications.
Williamson, Hilary, and Rohifs, Rabine, "The user interface design process", *Computer Message Systems*, North-Holland, N.Y., 1981, pp. 427-445. Using a message system as an example, they give concrete steps for designing a user interface. Steps given are setting goals, creating a functional model, researching scenarios of use, limiting dimensions of use, defining feature sets, developing personalization, writing user documentation, and finalizing the interaction language.

Xerox Corporation, "Mail Tools", *XDE Xerox Development Environment*, (formerly Laurel), pp. 30-1 to 30-18. This is the Xerox development environment interface for mail. There is a single window showing the "current" message, another window showing a summary of all messages, and a command window. The tool allows for three kinds of messages: short notes of plain ASCII, longer notes of text which may require conversion, and short notes (cover sheet) with attachments.

Xerox Corporation, *Star Mail User Guide*. This is mail for the Star office automation system. Mail is integrated into the desktop by an inbox and outbox. The Inbox changes icon when mail is available. Files are sent by dragging any file or folder into outbox. Messages are a "cover sheet" of fixed header fields and short note with an optional document or folder. Because this is integrated into the desktop, received files can be read as any other file.
Communications Interface

Birrell, Andrew D. and Nelson, Bruce Jay, *Implementing Remote Procedure Calls*, ACM Transactions on Computer Systems, v.2 n.1 (February 1984), pp. 39-59. A paradigm for communicating over a network is to think of network actions as remote procedure calls. In the first step of a remote procedure call, the call and parameters are passed to the local RPC handler. This handler knows the types expected for each call, converts these types into a Xerox RPC standard, and stuffs them into a packet. The Packet is sent to the server containing the call and the converted parameters. Upon delivery, the server RPC handler converts the parameters into the appropriate local format and passes the parameters to the appropriate routine. The called procedure runs, completes, and returns any results to the server RPC handler. The server RPC handler converts the results into the standard format, and sends a return packet. The local RPC handler receives this packet, and decodes and returns the result to the original calling procedure. An additional very important feature was that at link time, type checking of parameters was done over the network in order to validate RPCs. The interfaces for the RPC handlers on both ends are automatically generated by a program.

Bochmann, Gregor V., and Pickman, John R., A methodology for the specification of a message transport system, *Computer Message Systems*, North-Holland, N.Y., 1981, pp. 221-232. The design of a message system can be split into two pieces. There is a section for the User interface and another for the Transport interface. Example primitives are given for the interface between the two sections.

Lantz, Keith A., "Uniform Interfaces for Distributed Systems", TR63, Doctoral Thesis, University of Rochester, 1980. The user interface should be separated from the service that the program provides. Example of a distributed computer environment with resource server and multi window virtual terminals. RIGITS design and RITS implementations are described.

Motolese, E., Fondi, R., Salvadori, E., and Vignali, M., "MAIL2K: A computer mail package for local applications", *Computer Message Systems*, North-Holland, N.Y., 1981, pp. 353-359. Messages are links (possibly over a network) to the real file. These links are in both the user's inbox and outbox. When a message is to be read, then the message is retrieved.

Postel, J., "Internet Message Protocol", RFC 759, USC/Information Sciences Institute, August 1980. This is a protocol for sending messages from a user agent through message processing modules to a destination. Deliver, Ack, Cancel, and Probe (address) message types are discussed.

Schroeder, Michael D., Birrel, Andrew D., Needham, Roger M., *Experience with Grapevine: The growth of a Distributed System*, ACM Transactions on Computer Systems, v.2 n.1 (February 1984), pp. 3-23. Grapevine is the distributed messaging system used internally (and now externally as NS mail server),
Experience with grapevine showed that some distribution lists are a constant fraction of entire community (interest groups). It was difficult to manage a list that large at a single server, so software was changed to accept wildcards. System was designed to forward messages to workstations, not to store messages, so an increase in people just reading messages from terminals caused file overloads. Also, the system was designed to read sequentially and not randomly, so was very inefficient when used by terminals.

Terry, Douglas B., and Adler, Steve, "COSIE Communications subsystem", ACM Transactions of Office Information Systems, April 1984, pp. 79-85. A messaging system for a network with multiple heterogeneous hosts was developed. The user can log onto any host and get messages. The system is self optimizing because the user's mailbox follows the movements of the user.

Xerox Corporation: The Remote Procedure Call Protocol, XSIS 038112, December 1981, Xerox, Stamford, CT. This lists the details of the RPC protocol. The different layers are message, object, and block. Messages can be either call, returns, or aborts. Valid data type are boolean, string, cardinal. Blocks are groups of data bytes.