

An Overview of the Andrew Message System

A Portable, Distributed System for Multi-media Electronic Communication

Jonathan Rosenberg
Craig F. Everhart
Nathaniel S. Borenstein

Information Technology Center
Carnegie Mellon University
Pittsburgh, PA 15213

July 1987

1. Introduction

This paper provides an overview of the Andrew Message System, which is in operation within the Andrew project at Carnegie Mellon University. The Andrew environment currently consists of 300 high-function workstations (typified by the IBM RT-PC) each running Berkeley Unix and attached to a large campus-wide network. A central file system provides transparently the appearance of a large, monolithic Unix file system. In addition, there are approximately 600 IBM PC's and 300 (University-owned) Apple Macintoshes that may also participate in the network.

The Andrew Message System (often referred to in this paper as the *AMS*) is a suite of programs that provides powerful mechanisms for viewing, creating and manipulating multi-media mail and bulletin board messages. The AMS is usable from both high-function Andrew workstations and low-end workstations, such as IBM PC's and Macintoshes. This paper discusses our goals in designing the message system, the primary parts of the system, some of our design decisions and a number of the problems we encountered implementing such an ambitious system in our environment. In addition, we present some directions for future work and some statistics about our system.

2. Prior Work

Although space does not permit a complete survey of previous work on message systems, a few efforts influenced our design so strongly that they should be mentioned. The Grapevine system [21, 3] first demonstrated the feasibility and utility of truly distributed electronic message systems. Malone's work on the Information Lens [15] has stimulated our interest in mechanisms for dealing with information flood, and indeed we hope to implement some of Malone's ideas in future versions of the AMS (section 6.2).

Our ideas about user interfaces have been shaped by a succession of mail and bulletin board systems, most notably TOPS-10 RdMail [13], various Emacs-based message systems, earlier Andrew systems for mail and bulletin boards [4] and interfaces to the Unix Net-

news system [22, 10]. Our passion for an integrated communication environment with a coherent and clean design can be traced in large part to personal experience with communication systems lacking in these aspects [6].

Other proposals have been made for distributed mail systems. Our design of a remote protocol for message system clients may seem similar to the Post Office Protocol [7]. The similarity is superficial, however, as the Post Office Protocol (POP2) server merely provides a method for retrieving and deleting mail from a remote host. POP2 contains no support for error recovery, bulletin boards, message transmission or message database manipulation. Our more complicated remote messaging protocol provides support for all of these functions and more (section 5.3.1). Pcmail [9], a distributed mail system done at MIT, defines a still higher-level protocol for communication between a user's agent and a remote mail repository. Pcmail allows considerable separation between the user agent and the repository, and is designed to provide reasonable service even when the user agent machine is only infrequently connected to the repository. This design requires user agents to maintain local state, though, and to be able to resynchronize their local state with the repository's state. The facilities available to the Pcmail user do not include multi-media mail or as comprehensive a set of information sources as are provided by the AMS

3. Andrew

The Andrew project is a joint venture of IBM and Carnegie Mellon University, the goal of which is to produce a suitable working environment for academic use of computers. The project is described in detail in the paper by Morris et. al. [16], but a few key points are noted here.

In Andrew, each user works on a high-function workstation (currently an IBM RT, Sun 2, Sun 3 or Dec MicroVAX) with 2 to 4 MB of RAM, a 1 MegaPixel bitmap display, a mouse and 40-70 MB of fixed disk local storage. Each workstation is running Berkeley Unix (or its equivalent) and is connected to a campus-wide network over which it can talk to several dedicated file server machines. Running in this environment are two basic components that make up Andrew: VIRTUE and VICE.

VIRTUE is the user interface portion, which runs on the workstation, and includes a window manager and a multi-media editor subroutine library (known as the *base editor library*). Use of the base editor library provides an easy methodology for manipulating multi-media

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the ACM copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Association for Computing Machinery. To copy otherwise, or to republish, requires a fee and/or specific permission.

documents. In addition, VIRTUE provides a number of application programs (a text editor, for example) that exploit the facilities offered by the base editor library.

VICE (purported to be an acronym for Vast Integrated Computing Environment) is the central file system [20]; it emulates a Unix file system, so that as users move from one workstation to another, their picture of the file system remains consistent. In VICE a client process, called Venus, runs on each workstation and acts as the user's agent in making requests of the central file system. Small modifications to the Unix kernel allow it to route remote file requests to Venus. Venus then uses a remote procedure call protocol to make requests of a server process executing on a dedicated file server.

VICE works by performing *whole file transfer*: when a file is referenced from a workstation, Venus transfers the entire file from the remote file server into a cache on the workstation's local file system. From that point on, the file is treated as a local file by the workstation operating system. Venus may be told to invalidate the cached file in the event that the remote file system has changed the copy of the file stored there. For the purposes of writing, the client process treats the local disk as a write-through cache.

The use of whole file transfer has some advantages over the alternative scheme of executing remote file read and write operations. First, once the file is transferred, remaining accesses are as fast as using a local file system. In addition, whole file transfer means that the unit of granularity in VICE is the file. This allows VICE to notify clients efficiently of invalid cache entries; a client need only be notified when an entire file is stored, not when individual records change.

Of course, there are disadvantages to whole file transfer. The most glaring problem, and the one that affects people the most, is that an entire file is transferred even if only a small portion is going to be read. Studies have shown that this is not generally a problem because most applications read all the way through a file [19]. It is clear, however, that whole file transfer is entirely inappropriate for database applications. Many of the functions provided by the AMS are database applications, and discussions of how the design of VICE affected our work will be found throughout this paper.

To provide flexible security for files, VICE allows an *access list* to be attached to each directory. These lists allow selective access to be given or denied to individual users or groups of users. The access rights allowed are read, lookup (ability to see the file names in a directory), write, delete, insert, lock (ability to lock files in the directory) and administer access.

Both major parts of Andrew have strongly influenced the design of the Andrew Message System. Each has made some parts of the system easier and some parts more difficult. VIRTUE has, through the base editor library, made it almost trivial to deal with multi-media messages,¹ but has thus introduced serious complexities into the manner in which messages are sent and received to non-Andrew systems. VICE has made it easy to create a message database that is entirely location-independent, but has introduced, by its distributed nature, new failure conditions neither expected nor dealt with

robustly by software written for typical, standalone systems. In particular, the file system conceptually simplified the mail delivery mechanism, but at the same time mandated a complete replacement of the existing mail delivery programs.

The stated objective of the Andrew project was to produce an effective working environment for high-function workstations. One of the goals of the Andrew Message System was to produce software that was usable from such lower-functionality machines as IBM PC's and Apple Macintoshes, and this required that our design be more general than Andrew's, and in particular that our communication mechanisms, though based on VICE, be more general and more portable than VICE.

4. Goals of The Andrew Message System

The Andrew Message System is an ongoing project with the goal of producing a production-quality electronic communication environment with several types of functionality that hitherto either have not been provided by electronic message systems or have been found only in experimental systems. To accomplish this, we set the following subgoals:

Reliability: As users grow to rely on electronic communication, they come to expect and demand that the underlying transport systems never lose their messages. Reliability is, therefore, of the utmost importance in an electronic messaging system. While there are many reliable message systems, the task of constructing such a system in the Andrew environment was especially challenging.

Machine and location independence: The AMS should allow users to read mail and bulletin boards from virtually any kind of workstation, transparently preserving user profile information to maintain a consistent message system state.

Integrated message database: The AMS should treat mail and bulletin boards uniformly as consisting of *messages*, allowing users to manipulate both with a single interface. Thus, we can provide a small set of tools that allow the manipulation of many kinds of information.

Separation of interface from functionality: The AMS architecture should make it easy to support multiple user interfaces while preserving for each the highest functionality.

Support for multi-media communication: The AMS should support messages that include formatted text, vector graphics, raster images, equations and other multi-media objects. Of course, some of these messages will look their best only when viewed on more powerful displays.

Support for coping with information flood: There should be mechanisms in the AMS for dealing with the flood of information that increasingly overwhelms users of large electronic communication systems. The AMS should avoid performance degradation in the face of a large volume of messages from diverse sites on various networks.

Flexible Architecture: The AMS should allow for easy expansion to include various kinds of functions not yet foreseen or implemented. We believe this is accomplished by having an open-ended architec-

¹But, see section 6.1.

ture that provides high-level manipulations of the message database as one of its services.

Flexible addressing: The AMS should allow messages to be addressed using a variety of user-friendly address forms. A user should be able to specify a recipient by user id, by name, or even by making a best guess at the user's name.

5. Parts of the Message System

The Andrew Message System consists of several separable components: the message delivery system, the white pages, the message database and message server, and the user interface programs. The *delivery system* is responsible for accepting new messages for delivery and moving them through the various delivery queues and into users' in-boxes via a collection of daemons. The *white pages* is a database and access mechanism that provides sophisticated mappings from user-specified names to final mail destinations. The *message database* is the entire collection of public and private messages. The *message server* provides a high-level program interface to these underlying components: it is the procedural interface to the message database and also provides interfaces to the validation of mail destinations (via the white pages) and to sending and receiving mail (via the delivery system). The *user interface programs* are clients of the message server (which may execute on a different machine than the message server); their task is to present the capabilities of the message server to users in a way tailored to the capabilities of their environment.

5.1. The Delivery System

The delivery system is a set of programs responsible for many of the steps involved in delivering a message. The responsibilities of the delivery system include

- acceptance of messages for delivery to and from non-Andrew systems
- construction and return of error messages to originators of messages that were victims of failed delivery
- mapping from user-specified addresses to a list of recipients
- acceptance of messages for delivery from Andrew applications (via the *dropoff interface*)
- document format translation between Andrew and non-Andrew systems

It is our belief that the most important attribute of a delivery system is reliability. For the Andrew delivery system, we define reliability as the successful *disposition* of a message at the following points:

1. When a message is accepted from another system; and
2. When an invocation of the dropoff interface succeeds.

Furthermore, a message is successfully *disposed* of when, for each recipient: the message is successfully delivered or an appropriate error message has been constructed and delivered to the originator. For Andrew recipients, successful delivery of a message consists of storing a copy of the message into the recipient's mailbox. Successful delivery to a non-Andrew system consists of transferring the message to the remote system's mail transfer agent.

While there is nothing special about this definition of reliability, there are special design and implementation problems on Andrew due to the semantics of the VICE file system. The primary complication is the fact that a remote, distributed file system provides new error modes: in particular, *temporary errors*, which are errors that will eventually succeed if retried. On a standard timesharing system with a local file system, an application program, for all practical purposes cannot see a temporary error during a file operation. With VICE, however, these kinds of errors are relatively common. They may be caused by the temporary outage of a remote file server or a network error.

As a concrete example of this, consider the delivery of a message to a user's mailbox, which consists of creating a file in a designated directory. On a local file system an attempt to open a new file can result in success or in one of a small number of permanent failure modes: *permission denied*, *user over quota* or *hardware failure*, for example. In the case of such a permanent failure, the appropriate response is to abort the delivery attempt and reject the message back to the originator. On Andrew, we have the additional possibility of a new failure mode: the file open failed because the remote server that would hold the file was down. In this case it is not appropriate to abort the delivery attempt; instead the message should be held and delivery attempted again.

Typically, code that performs reliable file system operations is riddled with assumptions about possible failure modes. The introduction of new modes by VICE has necessitated the creation of new delivery software for the Andrew environment. We have managed, however, to retain the use of `sendmail [1]` as our SMTP user and server by modifying it and restricting its execution environment.

Many messages constructed on Andrew take advantage of the multi-media capability afforded by the base editor library (section 3). The external representation of such messages contains ASCII encodings of fonts, structure information and multi-media objects. This datastream allows users of the advanced interface (section 5.4.2) to view all of the objects in the original message. When a formatted message is transmitted outside of Andrew, however, the datastream format may make the message unreadable. To avoid this problem, the *delivery system* formats the message for a "standard" display device: all font information is removed, lines are formatted to be 80 characters wide, text is centered as appropriate and other reasonable transformations are performed. Of course, attempting to format certain kinds of objects for an arbitrary display device--graphics, raster and equations, for example--is impractical. Although we have not actually had to deal with these objects yet (section 6.1) we anticipate that such objects will simply be replaced by an "object omitted" marker.

Other sites are running Andrew and, thus, have the ability to recognize the base editor datastream. It would be nice if messages transmitted to these systems retained the formatting information. For this reason, the delivery system maintains a list of external sites that are running Andrew and transmits messages to these systems untouched. This is not a perfect solution: some users at the remote site may be using an interface without graphics capabilities, or the recipient may have his mail forwarded to a system not running Andrew. This is a difficult problem and the ideal solution depends on the *widespread acceptance of mail and multi-media document standards* [8, 11].

The delivery system supports some special addressing services by interpreting specially tagged address forms. These addresses have the syntax

`+<keyword>+<args>`

This is an open-ended notation that allows us to add keywords and new functionality as the need arises. Currently the delivery system supports two special services. The special form

`+dist+<file name>`

will cause the delivery system to treat the contents of `<file name>` as a distribution list. Besides a list of addresses, the file contains information specifying the address to which delivery errors should be sent. The address

`+dir-insert+<directory name>`

tells the delivery system to insert the message into the specified directory, just as if it were a mailbox directory.

5.2. The White Pages

The white pages facility contains both a database of recognized mail addresses and a library of procedures for mapping name probes to those addresses. This facility is used both by the delivery system (section 5.1) and by message composition (section 5.4). The delivery system uses the white pages to map the destination names given with a message to a list of mail addresses. During message composition an interface uses exactly the same facility to validate the destination names given by a user. The validation occurs interactively, so the user may correct addressing errors immediately rather than having to wait for a rejection notice. Using the same procedure in both places guarantees that a consistent interpretation is placed on all addresses.

The white pages facility supports one of the primary goals for the AMS: flexible naming of mail destinations. We wanted to allow people to address mail to Andrew users by specifying incomplete forms of users' names--their best guesses--as well as by specifying a unique user id. For example, we allow user "Jello Biafra," with user id "jb34", to be addressed in any of the following ways:

```
jb34@andrew.cmu.edu
Jello.Biafra@andrew.cmu.edu
J.Biafra@andrew.cmu.edu
```

and even

```
Gell.Byafro@andrew.cmu.edu
```

We assume that these addresses unambiguously identify Jello Biafra. The last form of addressing is permissible because we are currently installing heuristics for recognizing many accidental misspellings of user names.

The procedure that does the lookup in the white pages returns an indication of how many matches there were for a name and how flexible it needed to be in order to find the matches. Thus, when the delivery system uses the white pages to look up a name, if the name turns out to be ambiguous, the delivery system uses the white pages to compose an error message to the sender that lists the names that matched. Also, if a name turns out to be only a heuristic match, the delivery system can choose to send an advisory note back to the sender, or even to reject the delivery attempt completely and return an error message. Correspondingly, when an interface validates an

address given by the user, the white pages may indicate that the address matches many possibilities, or is only a heuristic match for a mail destination. In such cases, the message composition system can ask the user either to choose from a list of possible matches to the given name or to confirm or reject the result of the heuristic match. This facility has proven to be quite useful with a large system such as Andrew (currently with over 4700 users); the support for sending to parts of names encourages people to attempt abbreviations of their correspondents' names. Immediate validation of mail destination addresses at message composition time grants users the freedom to experiment.

The white pages stores information about users and special mailboxes; the information includes users' names, possible aliases, user ids, home directories (used to find the mail in-box directory) and forwarding addresses. The information is gathered from many sources, including the list of accounts (`/etc/passwd`) and a list of special mailboxes.

Building the white pages database on VICE has been a challenge. While it might be reasonable to store the entire database as a single Unix file, it is not reasonable to store it as a single file in VICE. This is because VICE insists that an entire file be transferred to the workstation to read even a small piece. Secondly, we have supported the Unix convention wherein users establish a mail forwarding address by creating a file named `".forward"` in their home directories. In order to keep the white pages up to date, a daemon must periodically look for changes to all users' `.forward` files, and must be able to distinguish and to tolerate temporary inabilities to examine users' home directories.

While the white pages database can grow to be very large, clients generally need to reference only a small portion. This locality of reference has allowed us to install a large database in VICE without requiring the transfer of large files to workstations. The B-tree discipline was designed to fit large amounts of data into a collection of fixed-size nodes, typically pages on a disk. We built a B-tree representation that uses a collection of VICE files as nodes, so that no file to be transferred need be larger than a reasonable size (currently up to 40,000 bytes). Our B-tree discipline supports concurrent reading and updating, using the B^{ink}-tree variant described by Lehman and Yao [14], in which readers need do no locking.

We store records describing users and indices to those records in the same B-tree, so that locating a user generally requires fetching only one or two leaves of the tree. This representation of the white pages has recently replaced our initial, interim representation, which used an existing facility that stored the database as a single file. The size of that file had grown to over one megabyte, and it was taking so long for addresses to be validated that our users required ways to circumvent validation. With the new representation, performance is much better, and we are able to remove our circumvention mechanisms.

Every night a daemon verifies the white pages database, incorporating new accounts, removing deleted accounts and checking for changes to the mail forwarding addresses in users' `.forward` files. It has been crucial that this verification process be able to tolerate the temporary unavailability of a user's home directory and files, even though the verification process cannot then know whether the user had a `.forward` file, much less what its contents would be. If the

verification process detects this temporary unavailability, it leaves the old mail forwarding information in place, with the expectation that having old information is better than having none. The initial state for a new account is to have a distinguished value **unknown** as its forwarding address. If the delivery system tries to deliver mail to an account whose forwarding address still has this distinguished value, it will attempt at that point to read the `.forward` file.

Eventually we will provide users with ways of updating parts of the white pages on-line. Users and administrators will send formatted messages to a designated address, and a daemon will carry out those requests, after verifying that the sender has appropriate permission. Once this mechanism is in place, we will no longer have to scan home directories looking for `.forward` files.

5.3. The Message Database

The *message database* is a hierarchical collection of all of the messages that may be manipulated by the AMS. The database is represented as a forest of directed acyclic graphs, similar in structure to the Unix file system. Each node is either a *message directory* or a message. A message directory may contain messages and other message directories. All of the nodes are stored in VICE so that although the database is distributed, it appears monolithic to users and application programs. Figure 5-1 presents a simplified view of the Andrew message database.

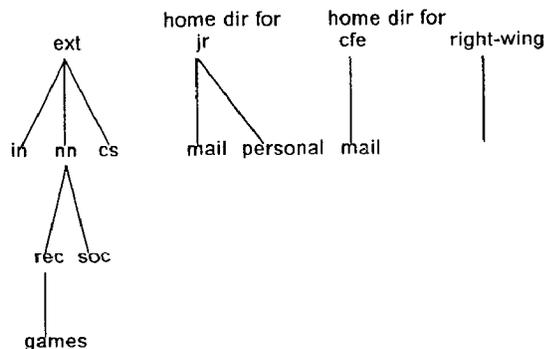


Figure 5-1: The Message Database

There are several interesting things to note about this picture. Some of the message directories are public and are stored in directories owned by Andrew administration. For example, the top-level message directories `ext` is public and centrally administered. Other directories are owned and administered by users: for example, personal mail directories and the `right-wing` directory.

Although the AMS is aware of the public message directories, and by convention can determine the location of personal mail directories, there is no root that can be used to find all the nodes of the message database. This means that the message database access routines (section 5.3.1) need not have a priori knowledge of the complete structure of the database. Users can create and maintain their own message directories without needing to register them in any manner with a central authority. For example, the `right-wing` message directory could have been created by a group of interested users without intervention by any Andrew administrators.

This ability provides great flexibility in the AMS by making it easy for any user to create and maintain a bulletin board. For convenience, a user may ask to have a bulletin board registered and its name will then be placed in a public, known location. Although this will make the name of the bulletin board publicly visible, it does not imply that anyone may read or post to this bulletin board. Because the bulletin board is stored as a VICE directory, the directory access list may be used to control access to the bulletin board.

A message directory is implemented as a VICE directory containing a distinguished file plus nested subdirectories and files containing the text of messages. This distinguished file contains administrative information as well as a structured record for each message in the directory. These records, known as *snapshots*, are of fixed size and allow rapid access to useful information about the messages. In particular, the snapshot for each message contains a time stamp and a condensed version of the header information from the message. The time stamp is used for ordering the messages and the condensed header information by user interfaces (section 5.4) to present summary information about a message. The user may then choose whether to view the entire message by inspecting this summary.

5.3.1. The Message Server

One particular goal we had for the AMS--to have the system available on the widest possible range of machines--has played a major role in its architecture. In order to make the system available on low-end machines, such as the IBM PC and Apple Macintosh, it was necessary to segment the functionality of the system into two major parts: the part that has access to the information in the database and the part that interacts with the user. The former, which we call the *message server* (MS), must run on a machine with full access to the message database as stored in the VICE file system. The latter, the user interface component, need only be able to talk to a message server via an agreed-upon mechanism.

The mechanism by which the message server and its clients communicate is called SNAP (for Simple Network Application Protocol), SNAP is a remote procedure call mechanism that was developed for the AMS and for use in connecting low-function workstations to the VICE file system [18]. SNAP runs on top of UDP [17] and supports sequencing, encryption and segmenting, thus providing a reliable packet protocol. The client code for SNAP (written in C) has been kept extremely simple to facilitate portability to multiple machines. To date, SNAP is running on the IBM RT-PC, Sun 2 and Sun 3 (under Berkeley Unix), the DEC MicroVAX (under Berkeley Unix and VMS), the IBM PC (under DOS) and the Macintosh.

The MS exports a subroutine interface that provide useful services for gaining access to and modifying the message database and for sending messages [5]. For example, the MS provides the following subroutines:

MS_SnapshotsSince

This routine is used to retrieve a set of snapshots from a designated message directory. The snapshots for the messages in the designated directory that have been entered since a specified date are returned. The message associated with each snapshot is marked with a unique id.

MS_GetPartialBody This routine is used to retrieve the body of a specified message, identified by its unique id. The message may be retrieved in fixed-size

chunks as specified by the client.

MS_CreateNewMessageDirectory

This routine is used to create a new message directory as a child of an existing message directory.

MS_SubmitMessage

This routine is used to submit a message for delivery to a list of specified addresses.

5.4. User Interfaces

The client-server design for access to the message database makes it relatively easy to create new user interfaces for the AMS to run on virtually any machine type. The implementor of an interface can program as if he were using a simple subroutine library. This allows him to ignore the intricacies of the message database regardless of the machine type on which the program will be executing. We provide a subroutine library, known as the CUILIB, that hides the complexities of the SNAP interface and allows the programmer to make local subroutine calls.

The ease of creating new interfaces is evidenced by the number of user interfaces that are available. The current interfaces of which we are aware are illustrated in figure 5-2.

	Andrew	VAX/VMS	PC
Messages	X		
CUI	X	X	X
PCmsgs	X		X
Batmail	X		

Figure 5-2: Current User Interfaces

In this figure, the first three interfaces were developed and are supported by the Andrew support and development staff. The *Batmail* interface is a popular, user developed and supported interface that runs within the EMACS text editor.

5.4.1. The CUI

The CUI (Common User Interface) is a simple, text-oriented interface that makes use of no graphics, screen or special input capabilities. The CUI is, thus, suitable to run on any terminal (even hard-copy) and in virtually any environment. The CUI has been ported to several machines (figure 5-2) and is, therefore, useful for the person wanting to learn only a single interface for use in many environments. An example of interaction with the CUI is shown in figure 5-3.

5.4.2. The Messages Program

Although the CUI provides full access to the AMS, it does not take advantage of the advanced features available on high-function Andrew workstations. For example, no use is made of the high-resolution graphics capabilities or of the mouse. For these reasons

```
CUI Version 3.30
CUI> update ext.nn.talk.origins
Checking ext.nn.talk.origins ...
1 7-Jul-87 Reality gold@bbn.com (508)
2 7-Jul-87 Purpose Bruce@sri (5558)
```

```
CUI READ> (Type '?' for help) [type]: type
From: jgold@cc6.bbn.com
Newsgroups: talk.origins
Subject: Re: The Nature of Reality
Date: 27 Jul 87 15:28:17 GMT
```

body omitted

```
CUI READ> (Type '?' for help) [next]: quit
CUI> quit
```

Figure 5-3: Using the CUI

we have implemented an interface that makes use of these features, known as **Messages**; an example of its display is shown in figure 5-4.

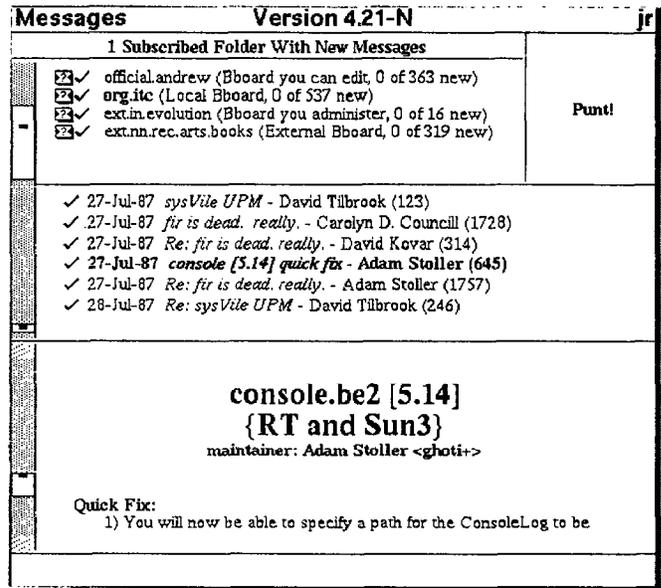


Figure 5-4: The Messages Interface

The Messages program is the interface of choice for most Andrew users. Considerable work has gone into the design of the interface that Messages presents; we have gone through several screen layouts and interaction styles before settling on one that seemed to please the widest number of users and styles of use. In fact, we currently offer two screen layouts, which differ in the placement of the message directory names, which appear in the top panel in figure 5-4. This technique of offering several alternative interfaces to users and collecting information on user reactions has been quite useful in the development of user interfaces on Andrew.

The Messages program is used to manipulate both mail and bulletin board messages. In fact, our initial design for the AMS included no distinction between personal mail and bulletin boards. While this may seem just plain wrong, consider that personal mail and bulletin boards are just two extreme points in a two-dimensional space of message directories. The axes are *the number of users who may read messages on this message directory* and *the number of users who may post messages on this directory*. Personal mail is a directory to which any user can post messages, but which only one user

can read. On the other hand, a public bulletin board is a directory to which any user can post and which any user can read.

There are other reasonable, useful points in this space. Consider a message directory that is readable by any user but only postable by a small number of designated users. This corresponds to an "official" bulletin board: any user may read messages and be assured that they were posted by authorized users. Another useful message directory is one that is postable by any user but may only be read by a small number of users. This kind of message directory could be used by a manager for his personal mail as it allows his secretary to also read (and, possibly, manipulate) his mail. Figure 5-5 displays some of the directory types available in the AMS.

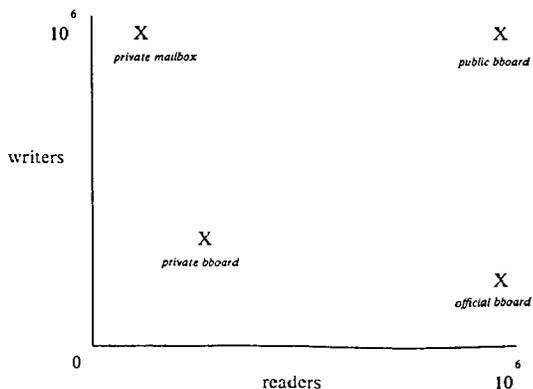


Figure 5-5: The Message Directory Space

The Messages program also supports an interface for composing and sending messages that uses the multi-media editing facilities of the base editor library (section 3). The user thus has access to a multi-media editor interface that is standardized across many Andrew applications.

The message composition interface takes full advantage of the flexible addressing afforded by the delivery system (section 5.1). In particular, the interface performs on-line validation of all forms of addresses. This provides immediate feedback to the user about local addressing errors and avoids the "send, receive error, correct error, resend" cycle.

In addition, the Messages composition interface provides two new forms of addresses: personal macros and bulletin board names. A user may maintain a list of personal macros in a designated file that is read by the Messages program. The appearance of one of these macros in an address list is expanded into the specified list of addresses. This is convenient for specifying commonly-used groups of users or for abbreviating long addresses. An example of such a personal file might be

```
group      cfe, ghoti, nsb, jr
mtg        group, mc35, KFS@vma.cc.cmu.edu
ams-folks  +dist+postman/ams-folks.dl
```

The macro expansion may reference other macros, or any other form of address.

Posting a message to a bulletin board is done simply by using the name of the bulletin board as a destination. For example, a message may be posted to the Netnews message group **rec.arts.books** by addressing the message to **ext.nn.rec.arts.books** (the Andrew name of the bulletin board). This form of addressing works regardless of the method of redistribution for the bulletin board. For example, messages addressed to Netnews are automatically routed to a daemon that transmits the message via the Netnews protocol [12]. This form of addressing allows a message to be sent to both users and bulletin boards.

Section 5.3 discusses the fact that users can create new private message directory trees without the need for intervention from system programmers. In addition, at designated places in the public database hierarchy it is possible for users to create a new message directory as a child of an existing directory. For example, referring once again to figure 5-1, it is possible for a user to create a new message directory under the node **andrew.market**. This would be useful, for example, if many postings on this bboard were about the sale of cars. Any interested user could decide that cars for sale warranted a bulletin board of their own, so as not to clutter the market bulletin board. This user could then create the new bulletin board simply by addressing a message to **andrew.market.cars**.

5.5. The PCmsgs Program

Although the CUI is available on IBM PC's, another interface that takes advantage of the display capabilities is available. This interface is known as **PCmsgs** and, like the CUI uses the CUILIB (section 5.4). In the same manner as Messages, PCmsgs presents displays listing message directories and message snapshots. Unlike Messages, which uses the mouse for input, users of PCmsgs maneuver around the screen using keyboard commands. Figure 5-6 shows a PCmsgs screen that displays the snapshots of the user's new mail.

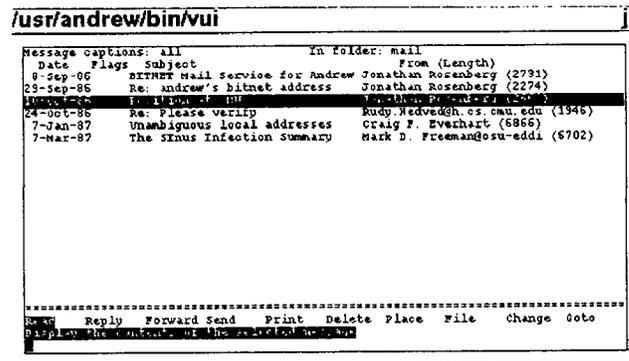


Figure 5-6: The PCmsgs Interface

Although PCmsgs was developed initially for the IBM PC (section 5.5), we have built a version of it that uses the Unix termcap interface [2] and, thus, runs on any display device that is supported by the user's particular Unix system. This version is especially useful when dialing in to Andrew.

6. Future Development

Although the Andrew Message System is already a rather sophisticated messaging system, it can be viewed as just the beginning. The delivery system, white pages, message database and the user interfaces can be thought of as the tools on which we might build an *information utility*. An information utility would provide users with access to a large number of diverse databases and access methods. Besides electronic bulletin boards and mail, there might be databases containing the full texts of popular newspapers dictionaries and encyclopedias.

Designing, integrating and implementing such databases and associated access mechanisms is a difficult task. The job is even more difficult in a distributed system like Andrew. The most serious problem on Andrew is that the VICE file system, with its whole file transfer paradigm (section 3), was not designed to support large databases. An efficient implementation for general database access on Andrew is an open question.

The proposed information utility contains a wide assortment of data with different timeliness constraints and modes of use. We would expect that there would be a corresponding need for different forms of user interfaces. While the bulletin board paradigm is appropriate for some databases, it is not clear that it is appropriate for viewing a newspaper. We need to investigate and experiment with new forms of interfaces.

Although we are looking forward to progressing towards this utopian information utility, more work remains to be done on the current AMS. The remainder of this section provides brief discussions of some of the most important tasks that we will undertake in the near future.

6.1. Integration of the Multi-media Editor

Several times in this paper we have mentioned the multi-media capabilities of the AMS. Any Andrew application can obtain these multi-media capabilities by using the base editor library. The base editor library is a suite of subroutines that provide an application with the ability to create, view, edit, print, read and write multi-media documents. These documents may contain text (including multiple fonts and complex structure, such as headings, chapters and paragraphs), vector graphics, raster images, equations and spread sheets. In fact, the base editor architecture is open ended and new multi-media objects may be added at any time. An example of a multi-media document, displayed using a base editor library application, is shown in figure 6-1.

While this picture is rosy, the truth is that the multi-media version of the base editor library is only just becoming available for use by applications. The current Messages interface uses an earlier version of the base editor library that supports only text (albeit with multiple fonts and complex structure). Integration of the Messages program with the new base editor library is underway. We expect to have a working multi-media version of Messages by October of this year.

6.2. Bulletin Board Reorganization--Real and Virtual

We have several plans for extending the services that the AMS provides for its users. We expect to complete a reorganization of our tree of bulletin boards this summer. The reorganization will not only

`/cmu/itc/ajp/ui/insets/bxdemo/zip_more` jr

Picture-figures, the width/height is arbitrarily set to 256 each way.)

Here follows a *file-reference* (ie, the source document contains a fully qualified file-name) to that famous, complex graphic `usa.zip`:



↑Now we have an *embedded* Zip-stream  (ie, the Zip ASCII characters are right here *and* within the sentence).

Figure 6-1: A Multi-media Document

clarify the position of the Andrew community on our campus and in the world at large, but will also allow departments and other organizations to publish their private bulletin boards in well-known locations. Also, we are implementing an extension language (FLAMES, the Filtering Language for the Andrew Message System) in which users will be able to compose calls to the message server without having to program in C. Perhaps our most novel extension will be an adaptation and extension of some of the work done in Malone's Information Lens system [15]--in particular, the work done by its "anyone server", in which users tell the system the kind of public messages that they would like to have sent to them. We expect to support a comparable service for a much larger user community by building on the services available in the extension language.

The goal of the message server extension language, FLAMES, is to allow message server clients to specify complex operations, including powerful database manipulations, that can be executed completely within the message server. Interface programs will have access to database manipulation primitives without having to re-code them for each new interface, and the database manipulations can take place in the message server rather than in the machine running the interface program. The advantage of moving complex computations to the message server host is that the message server is guaranteed to run on a high-function processor, connected over fast communication lines to VICE, whereas interface programs may execute on limited machines or over slow communications links.

The extension language is being built with a simple Lisp syntax, and we envision having two interpreters: a simple interpreter running FLAMES within the message server itself, and a full Lisp system running extended FLAMES (X-FLAMES). X-FLAMES will be available as a client of the message server. All interface programs will be able to ask a message server to execute a command in FLAMES. More complicated operations, such as we anticipate providing as a system service, can be handled by invoking the X-FLAMES interpreter directly.

It is this latter ability that we expect to exploit in building a massive information lens. A user of the information lens posts a predicate specifying the kinds of messages that are of interest. For example, someone might want to read all messages that are about automobiles or are from a specific message source and are about California wines. The user might post a predicate like

```
(union
  (word-search "automobile")
  (word-search "autos")
  (word-search "cars")
  (intersection
    (word-search "California")
    (word-search "wine")
    (bboard "rec.food.drink"))))
```

This is a predicate that matches both messages that contain the words "automobile", "autos", or "cars", as well as messages from the bulletin board **rec.food.drink** containing both the word "wine" and the word "California".

In principle it would be possible for an interface program to find the set of all messages currently matching this predicate, then to subtract from that set the ones that the user has seen already, and finally to present to the user the new messages that match the predicate. This action would be too expensive to carry out today, but we hope to be able to support such actions by clever preprocessing performed by a special lens system application written in X-FLAMES. Our plan is to collect all the predicates posted by the lens users, to find common sub-expressions of these predicates (such as "(word-search "cars")") the results of which would accelerate the evaluation of later predicates. The lens system application would sift through all incoming messages and identify those messages that match the common sub-expressions. Suppose the Lens system application examined every incoming message destined for a public bulletin board and posted all messages that contained the word "automobiles" to a bulletin board named **lens.wordsearch.automobiles**. This preprocessing would make it more efficient to evaluate parts of predicates. If the first three word searches in the example above were maintained by the Lens system, evaluating the entire predicate would be fast indeed: a program would need to check for messages on only the three **lens.wordsearch** bulletin boards and the single bulletin board **rec.food.drink** before identifying the new messages of interest. With this kind of pre-indexing, in which the community of users influence the kinds of indices kept, we expect to be able to support the advanced information needs of a large community.

7. Statistics

This section contains a number of statistics related to Andrew and to the Andrew Message System. The figures are presented without commentary and are intended to give the reader a feel for the size of Andrew and its message system.

- 300 Andrew workstations
- 11,000 campus network access wall outlets
- 4700 Andrew accounts
- 17 gigabytes of storage available in VICE
- 1000 public bulletin boards

- 1 bulletin board post per minute (averaged over a 24 hour day)
 - 7 posts per hour received for ARPAnet bboards
 - 40 posts per hour received for Netnews bboards
 - 4 posts per hour received for local bboards
- 24 non-bboard messages per hour received from other sites
- 10 posts per day sent to Netnews bboards
- 34 messages per day sent to non-Andrew sites

Acknowledgements

Besides the authors, Adam Stoller is the fourth full-time member of the Andrew Message System group. Adam's primary job is to administer the AMS. This includes monitoring of the message system server machines (of which there are currently five), administration of the message database, answering users' requests and gripes (a never-ending and thankless task on Andrew) and other related problems that fall through the cracks. Without Adam, the Andrew Message System would not be as reliable, robust and responsive as it is today.

The message system would have been impossible to implement without the help of the entire Information Technology Center staff. In particular, we would like to thank the VICE file system group for being responsive to our needs and to the bugs we uncovered.

References

- [1] Eric Allman.
SENDMAIL -- An Internetwork Mail Router.
In *UNIX System Manager's Manual*. University of California, Berkeley, 1986.
- [2] Kenneth C. R. C. Arnold.
Screen Updating and Cursor Movement Optimization: A Library Package.
In *UNIX Programmer's Manual: Supplementary Documents*. University of California. Berkeley. 1984
- [3] Andrew D. Birrell, Roy Levin, Roger M. Needham and Michael D. Schroeder.
Grapevine: An Exercise in Distributed Computing.
Communications of the ACM 25(4):260-274, April, 1982.
- [4] Sandra J. Bond.
Module 4: The Andrew Mail and News Systems.
Information Technology Center, Carnegie Mellon University, Pittsburgh, PA, 1986.
- [5] Nathaniel S. Borenstein.
The Andrew Message System Server-Client Interface.
Internal documentation, Information Technology Center, Carnegie Mellon University, Pittsburgh, PA, November, 1986.
- [6] Nathaniel S. Borenstein.
A House of Cards: A History of the Inorganic Evolution of the CMU Bboard System Software.
Technical Report CMU-CS-85-152, Computer Science Department, Carnegie Mellon University, Pittsburgh, PA 15213, 1985.
- [7] M. Butler, J. Postel, D. Chase, J. Goldberger and J. K. Reynolds.
RFC 937: Post Office Protocol - Version 2.
USC, Information Sciences Institute, 1985.

- [8] Study Group VII.
Red Book, Volume VIII, Fascicle VIII.7: Data communication networks: message handling systems. Recommendations X.400-X.430.
CCITT, the International Telegraph and Telephone Consultative Committee, 1984.
- [9] David D. Clark and Mark L. Lambert.
RFC 993: PCMAIL: A Distributed Mail System for Personal Computers.
Massachusetts Institute of Technology, 1986.
- [10] Mark R. Horton.
RFC 850: Standard for Interchange of USENET Messages.
USC, Information Sciences Institute, 1983.
- [11] ISO TC97/SC18.
Information Processing - Text and Office Systems Office Document Architecture (ODA) and Interchange Format.
Technical Report ISO/DIS8613, International Organization for Standardization, June, 1987.
- [12] Brian Kantor and Phil Lapsley.
RFC 977: Network News Transfer Protocol.
UC San Diego, 1986.
- [13] David Alex Lamb.
RdMail Message Management System: User's Guide and Reference
Seventh edition, CMU Computer Science Department, Pittsburgh, PA, 1982.
- [14] Philip L. Lehman and S. Bing Yao.
Efficient Locking for Concurrent Operations on B-Trees.
ACM Transactions on Database Systems 6(4):650-670, December, 1981.
- [15] Thomas W. Malone, Kenneth R. Grant, Franklyn A. Turbak, Stephen A. Brobst and Michael D. Cohen.
Intelligent Information-Sharing Systems.
Communications of the ACM 30(5):390-402, May, 1987.
- [16] James H. Morris, Mahadev Satyanarayanan, Michael H. Conner, John H. Howard, David S. H. Rosenthal and F. Donelson Smith.
Andrew: A Distributed Personal Computing Environment.
Communications of the ACM 29(3):184-201, March, 1986.
- [17] J. Postel.
RFC 768: User Datagram Protocol
USC, Information Sciences Institute, 1980.
- [18] Larry K. Raper.
The CMU PC Server Project.
Technical Report CMU-ITC-051, Information Technology Center, Carnegie Mellon University, February, 1986.
- [19] M. Satyanarayanan.
A study of file sizes and lifetimes.
In *Proceedings of the 8th Symposium on Operating Systems Principles*. Asilomar, CA, December, 1981.
- [20] M. Satyanarayanan, John H. Howard, David A. Nichols, Robert N. Sidebotham, Alfred Z. Spector and Michael J. West.
The ITC Distributed File System: Principles and Design.
In *Proceedings of the 10th Symposium on Operating Systems Principles*. December, 1985.
- [21] Michael D. Schroeder, Andrew D. Birrell and Roger M. Needham.
Experience with Grapevine: The Growth of a Distributed System.
ACM Transactions on Computer Systems 2(1):3-23, February, 1984.
- [22] Larry Wall.
RN Manual Page.
1985.
RN is distributed via netnews; the author's mail address is lwall@sdcrdcf.UUCP.