Power, Ease of Use, and Cooperative Work in a Practical Multimedia Message System

Nathaniel S. Borenstein[1]
Chris A. Thyberg[2]

for publication in
The International Journal of Man Machine Studies
Special Issue on Computer Supported Cooperative Work and Groupware

Abstract/Summary

The "Messages" program, the high-end interface to the Andrew Message System (AMS), is a multimedia mail and bulletin board reading program that novices generally learn to use in less than an hour. Despite the initial simplicity, however, Messages is extremely powerful and manages to satisfy the needs of both experts and novices through a carefully evolved system of novice-oriented defaults, expert-oriented options, and a help system and option-setting facility designed to ease the transition from new user to sophisticated expert. The advanced features of the system facilitate types of cooperative work that are not possible with other mail or bulletin board systems, but which would also be impossible in large heterogeneous communities if the system were not so easily used by both novices and experts. A major example of such cooperative work is the Andrew Advisor system, a highly-evolved and sophisticated system that uses the AMS to solve the problems of distributed support for a very diverse user community in a heterogeneous computing environment. The evolution of the Advisor system and its uses of the AMS mechanisms are considered as a detailed example of the power and limitations of the AMS.

Introduction

This paper describes one notably successful user interface program for reading and sending mail and bulletin board messages, the "Messages" interface to the Andrew Message System. This system is currently in use at hundreds of sites, and at some sites its use has become virtually ubiquitous. In such environments, where its advanced features can be universally relied on at both ends of the communication, it has facilitated new kinds of computer-based cooperative activities. In this paper, we will describe the Messages program in order to understand the factors underlying its success, both its popularity with users and its effectiveness as a tool for cooperative work. In particular,

---

[1] This paper describes work carried out while the author worked at the Information Technology Center, Carnegie Mellon University, Pittsburgh, Pennsylvania. The author's current address is: Bellcore, Room MRE 2A274, 445 South Street, Morristown, NJ, 07960. The author's electronic mail address is nsb@thumper.bellcore.com.

[2] The author's address is: Room 3017, Hamburg Hall, Carnegie Mellon University, Pittsburgh, PA, 15213. The author's electronic mail address is cat+@andrew.cmu.edu.

we will focus on the question of how it manages to accommodate the diverse needs of novices and experts alike. Finally, we will look at an example of how the system has been successfully used by an independent group to support a rather complex form of cooperative work, the Andrew Advisor system.

Underlying Design Principles

A good user interface is, of course, always good news to the people who have to use it. All too often, however, it has proven difficult or impossible to determine, after the fact, what has made a user interface successful or popular. The lessons of popular user interfaces are often idiosyncratic and difficult to generalize, or just plain obscure, as noted in Borenstein & Gosling (1988). In the case of the Messages program, as with all others, a great deal of debate could be made over the reasons for its strengths and weaknesses, or indeed over the precise nature of those strengths and weaknesses. In this case, however, the program was initially built and subsequently remodeled a clear foundation of assumptions and beliefs about user interface technology, so that the end product may justifiably be viewed as the result of an experiment, an empirical application of one set of user interface design principles. We will make these principles explicit before describing the program itself.

The principles put forward here were not explicitly stated or committed to print prior to the Andrew project, but they were certainly strongly-held beliefs that were often expressed in conversation. One of the authors has recently produced an expanded attempt to enunciate these as general principles for user interface design (Borenstein (1990a)). In that book, arguments are made to justify the principles. Here, however, we will treat the principles as axioms, and will consider the resulting artifact, the Messages program, as empirical result of the application of these axioms. Or, to put it more simply, we describe the principles and the result, in the hope that the connection between the two will tend to support the validity of the basic design principles involved.

Assumption 1: The actual utility of applications that promise to support Computer-Supported Cooperative Work (CSCW) cannot be judged in the absence of a real user community. Any system, therefore, that claims to make a contribution to CSCW, but has no significant base of regular users, is making an empty or unverifiable claim.

Assumption 2: Usability is an essential prerequisite for any software system with a significant user interface component, which includes all systems to support cooperative work. Even in "research" systems, if the focus of the research is on doing something for end users, as it necessarily must be in all CSCW research, then a highly polished and usable interface is essential. The absence of such an interface will make it nearly impossible to obtain a realistic user community, and will thus necessarily skew any research results in such a way as to make it nearly impossible to evaluate the underlying ideas.

Assumption 3: In user interfaces, there is *no* fundamental trade-off between power, complexity, and usability. The most complex and powerful systems can also be the easiest to use, if designed properly, subject to ongoing, consciously evolutionary development.

Assumption 4: In a complex user interface, all defaults should be carefully tuned for the most common novice user responses and expectations.

Assumption 5: Powerful but potentially confusing user interface features should be turned off by default, so as to not conflict with novice learning.

Assumption 6: Mechanisms must be provided to ease the transition from novice to expert, especially in systems where powerful expert-oriented features are not made available without explicit user action to request them.

Assumption 7: Good user interfaces are grown and evolve. The most essential part of the design process is the evaluation of and improvement upon previous versions of the interface, based on feedback from and observation of real users of the system.

This paper views the Messages program as an uncontrolled field test of the above assumptions. The successes and failures of the system cannot be absolutely demonstrated to have resulted directly from these assumptions, but it is the authors' belief that a substantial connection does exist. At the very least, the principles provide the philosophical background against which the system should be understood.

Andrew and its Message System

Besides the philosophical background, there is also a technical background that must be understood in order to have a clear understanding of the Messages program. Messages was produced as a part of the Andrew project, about which a brief explanation is in order.

The Andrew Project (Morris, Satyanarayanan, Conner, Howard, Rosenthal, and Smith (1986), Morris (1988), Borenstein (1990b)) was a collaborative effort of IBM and the Information Technology Center at Carnegie Mellon University. The goal of the Andrew project was to build a realistic prototype of a university-wide distributed computing environment. That is, particular emphasis was paid to the needs of the academic and research communities. The success of that effort can be measured in part by the fact that the prototype has been taken up and is now fully supported by the University's central computing organizations.

As the project evolved, it concentrated on three main parts. The Andrew File System (Howard (1988), Howard, Kazar, Menees, Nichols, Satyanarayanan, Sidebotham, & West (1988), Kazar (1988), Kazar & Spector (1989)) is a distributed network file system designed to provide the illusion of a uniform central UNIX file system for a very large network (10,000 workstations was the design goal).[3] The Andrew Toolkit (Palay,

---

[3] The Andrew File System technology, AFS 3.0, is a product of Transarc Corporation.

Hansen, Kazar, Sherman, Wadlow, Neuendorffer, Stern, Bader, & Peters (1988), Borenstein (1990c)) is a window-system-independent programming library to support the development of user interface software. It currently supports a number of applications, including a multi-media editor that allows seamless editing of text, various kinds of graphics, and animations.

The third main piece of Andrew is the Andrew Message System, or AMS. The AMS, which makes heavy use of the file system and the toolkit, provides a large-scale mail and bulletin board system. It transparently supports messages which include text, pictures, animations, spreadsheets, equations, and hierarchical drawings, while also supporting "old-fashioned" text-only communication with low-end machines such as IBM PCs and with the rest of the electronic mail world. The Andrew Message System has, in recent years, become widely available; while the Carnegie Mellon installation is still the largest by some measures, there are other large Andrew sites, one of which has a bulletin board system at least twice as large as Carnegie Mellon's. This paper primarily reflects experience with the system at Carnegie Mellon, however, as that is where the system was developed, has been used for the longest time, and has been most readily observed by the authors.

There are many parts to the Andrew Message System, including several non-multimedia user interfaces for reading mail and bulletin board messages from low-end terminals and PCs. There are also several AMS subsystems that have relatively small user interface components, such as the distributed message delivery system. A detailed description of the Andrew Message System is beyond the scope of this paper and can be found elsewhere (Rosenberg, Everhart, & Borenstein (1987), Borenstein, Everhart, Rosenberg, & Stoller (1988), Borenstein & Thyberg (1988), Borenstein, Everhart, Rosenberg, & Stoller (1989)). This paper will concentrate on the high-end user interface, the "Messages" program, and on the manner in which it has proven to be particularly conducive to cooperative work.

Messages: The System Functionality

Although the AMS is a complex system made of many parts, to most users the term "AMS" is virtually synonymous with the Messages user interface program, which is all they actually see of the AMS. Messages presents a basic user interface that is quite similar to many other mail and bulletin board readers, easing the learning process for many users. Hidden behind the superficial similarity, however, is a wealth of powerful features that await the interested user.

The Messages Windows

Messages runs under any of several window management systems, the most common of which is the X11 window system from MIT (Scheifler & Gettys (1987). The program can open multiple windows on the screen, but typically the novice user is confronted with the single window shown in Figure 1, in which the screen is divided into several subwindows for message bodies, message "captions" (one-line summaries), and the names of message "folders" (collections or directories of messages, analogous to mail classes in some other systems).

Within this main window, the novice user can do everything one might need to do in the course of *reading* mail and bulletin board messages. The most common actions -- selecting a new message or folder -- are accomplished by pointing and clicking. Other actions, such as deleting messages, are available via the standard Andrew pop-up menu mechanism. For the novice user, there is never any reason to touch the keyboard in the course of reading messages.

To send a message, a user may either choose the "**Send Message**" menu item or one of the "***Reply***" menus. This will cause a new "messages-send" window to appear on the user’s screen, as pictured in Figure 2.

Multimedia features

A major area in which Messages offers more functionality than most mail and bulletin board systems is in the integrated manner in which it includes formatted text and multimedia objects. In Figure 3, for example, the user is reading a message that contains a picture within formatted text. It is important to note that users can read, print, and otherwise manipulate such messages with absolutely no knowledge about the multimedia system. Multimedia messages are fundamentally no different, from the user’s perspective, than any other messages in the system, and the user need learn nothing new in order to read most of them, and only a few new things in order to compose them

The multimedia capability of Messages has, perhaps not surprisingly, proven to be one of its most admired and successful features. Crucial to its success has been the fact that novices can receive and appreciate multimedia features with essentially no extra effort or learning. Also critical has been the ease with which new and casual users can master a subset of the multimedia authoring capabilities and still get substantial benefit from that subset. Nearly all Messages users quickly learn, for example, the ease and value of using multiple fonts within mail messages.

Active Message Features

Another aspect of Messages that has proven extremely useful and popular is a set of features known collectively as "active messages." These are a set of specialized message types that carry with them, in addition to a normal (and possibly multimedia) message body, information that directs a particular interaction with the user. For example, one type of active message is the "vote" message. Here special headers direct the user interface to ask the user a multiple choice question, the answer to which will be mailed to a designated address for collection and tabulation. Figure 4 shows a user reading a vote message. In addition to votes, the Andrew Message System supports four other types of active messages: return receipt requests, enclosures, folder subscription invitations, and redistribution notices. (See Borenstein, Everhart, Rosenberg, & Stoller (1989) for details on active messages).

As with multimedia messages, active messages require no special training to be of value to the receiver. For the receiver, they appear simply as messages that magically bring up dialog boxes and ask questions using mechanisms that are easily understood. The amount of expertise required to create an active message is also surprisingly small and is easily mastered by new users of the system.

It seems likely that the notion of "active messages" can be generalized substantially. This is the subject of one of the authors' recent research, Borenstein (1990d).

The FLAMES Message Filtering Language

The AMS provides an embedded LISP-like language called FLAMES (Filtering Language for the Andrew MEssage System) that can be used to automatically classify new mail when it arrives. By default, new mail is placed in an automatically-created folder called "mail." However, a FLAMES program can sort incoming mail by keywords, by sender, or by any other aspect of the mail message, and can automatically place mail in the correct folder. (It is important, however, that the user "subscribe" to any folders in which mail is placed automatically, or the system will not automatically show the user the new messages in those folders.) Indeed, a FLAMES program can even reject mail by returning it to its sender, or it can automatically process the mail and send out an answer. The most common use for personal FLAMES programs is to automatically sort new incoming messages into folders. Beyond this, however, several complex FLAMES-based applications have been developed, and the Advisor system, to be described later in this paper, relies heavily on FLAMES for message processing.

Private Bulletin Boards and New Bulletin Board Creation

The Andrew Message System supports a rich and flexible set of protection and configuration options that facilitate group communication. In particular, the protection mechanisms permit the creation of public bulletin boards, private bulletin boards (readable and postable only by members of a group), official bulletin boards (readable by all, postable only by a few), administrative and advisory bulletin boards (postable by all, readable by only a few), and various hybrids thereof. In addition, the protection mechanisms can be (and are) used to allow, for example, a secretary to read and process someone else's electronic mail. (Indeed, a secretary could create something like a magazine for an employer, containing only those pieces of the employer's mail that the secretary thought the employer would really want to see.) The rich protection options make it possible to use message "databases" in innovative ways, as will be illustrated later in this paper.

Customization Options

Most of the optional features that have been described are relatively easy to learn. Beyond this, however, the Messages program is radically customizable using mechanisms that require substantially more expertise. The Andrew Toolkit, on which Messages is based, provides several such mechanisms, on several levels. In particular, it includes an "init file" mechanism, which offers a simple macro facility for creating compound commands. For situations where such a simple facility is inadequate, the toolkit includes Ness, an extension language described in Hansen (1990), which allows fully programmable customizations and extensions to the behavior of AMS, as well as the creation of powerful interactive objects that can be sent and received with Messages.

Though these mechanisms are complex enough to require substantial time and expertise to master, they are sufficiently useful and accessible to have been used on many occasions to create customized or extended versions of the AMS for specialized purposes, one of which will be discussed at some length later in this paper.

Other Advanced Features

The AMS supports many other advanced features, too many to describe in detail here. These include:

-- Electronic "magazines" which allow one user to act as an "information filter" for many other users and thus reduce the problem of "information flood."

-- An unusually rich set of mechanisms for replying to messages.

-- Support for easily including excerpts from one message in another in an aesthetically pleasing way.

-- Heuristic validation of destination addresses.

-- A rich set of variants on the basic notion of "subscribing" to a message folder.

-- A large amount of functional support for manipulating message folders.

-- Mechanisms for marking groups of messages and manipulating them as a group.

Learning About and Using the Optional Features

As the Messages interface evolved, in every case where a choice had to be made between the needs of novices and the needs of experts, the default behavior of the program was targeted at novice users. The resulting program is undeniably easy for novices to use. For experts, the desire for extended functionality is accommodated through the use of options.

This is, in general, a tricky and risky enterprise, because there is really no difference between a non-existent feature and a feature that the expert doesn’t know about or can’t figure out how to use. In order to successfully meet the needs of experts, it was important to ensure that no major expertise would be required in order to use the expert-oriented features.

The most important mechanism by which this is accomplished in the Messages program is the "Set Options" interface. In any message-reading window, the user can choose the "**Set Options**" menu option. When this menu action is initiated, the display is altered, as shown in Figure 5. Here the contents of the "captions" area have been replaced with a scrollable list of user-settable options, and the "bodies" area now displays a scrollable set of option-related information, including interaction objects that can be used to actually change the options.

Using the "Set Options" interface, users can easily learn about and use a large number of sophisticated options. By the time they have exhausted the potential of this interface, they are already expert Messages users by any reasonable definition. Beyond this point, further customization is still possible using more complex mechanisms, as previously mentioned. Although the Andrew help system provides significant assistance to users who want to master these mechanisms, they remain significantly harder than the "Set Options" mechanism. Most users never even attempt to learn to use the other mechanisms, so it is important that the needs of the majority of sophisticated and expert users be satisfied by the use of "Set Options."

The Myth of The Power/Usability Tradeoff

There is a popular and widespread belief among programmers and end users alike that a fundamental tradeoff exists between easy-to-use, novice-oriented programs on the one hand and very powerful and customizable expert-oriented programs on the other. This belief persists in the absence of any really compelling evidence, and in spite of the existence of at least a few examples of programs that successfully "have it both ways."

Along with a handful of other programs, the Messages interface can be viewed as a proof-by-example of the fact that this is not a fundamental tradeoff. There is no reason *in principle* why an interface cannot meet the needs of both experts and novice users. Indeed, doing so is startlingly simple in theory, though exceedingly difficult in practice. Basically, only three things are required:

1. An easy-to-use, novice-oriented default interface.

2. A large set of powerful features and options that are not visible or enabled for new users.

3. A smooth, obvious, and easy-to-use mechanism by which users can gradually learn about the more advanced features.

Of course, all three of these things are much more easily said than done. In the case of Messages, these three things were successfully obtained only after a great deal of evolution, user testing, and independent evaluation. But it is important to understand that the popularity and success of the Messages interface was not attributable to any particular intuitive genius on the part of the builders, but rather to the process and environment in which the interface was developed.

The initial public releases of the Messages program, in particular, satisfied almost none of the users. Novices found the screen layout of the initial version, which mixed folder names and the new messages within each folder in a single scrollable text region, to be confusing and unintuitive. Experts, meanwhile, were frustrated by the many features that had been omitted in the name of usability (and also for expediency). In fact, the initial version was met with such hostility that it would have been reasonable to consider simply abandoning the whole project. The fact that the program was able to evolve into the popular interface described in this paper is indicative of the fact that something was done right in the process by which the system evolved.

The first salient feature of that evolutionary process is that it was long and painful. It took about four years of full-time programming work by one person, with additional work by many others at many points. Most of this time was spent trying to get a great number of details right. It is not at all obvious how the process could have been significantly streamlined. There just may be no substitute for sweat and hard work.

Another aspect of the evolution worth noting is that, from the second version on, the Messages program always had a large community of experienced users as well as a continuous influx of novice users (in the form of incoming freshman students at CMU). The expert users helped guarantee the continuing accretion of expert-oriented features, while the steady stream of new users ensured that the default settings would continue to be refined towards ease of use for novices.

Also crucial during this period was the fact that Messages captured the attention of a number of non-technical specialists who helped to guide its evolution. The Andrew project was able to hire, as consultants, a graphic designer to study the visual aspects of the program, technical writers to improve the documentation and interaction messages, and a human factors expert to study how novices and experts actually used the system and where they got stuck.

Most important, the Messages interface was able to evolve successfully because of the tenacity or stubbornness of many of the parties involved. The author bullheadedly proceeded from the assumption that nothing could possibly be wrong with the interface that couldn’t be fixed with enough work -- an attitude which, while it produced a good interface in the end, may well have produced a much bigger system than was strictly necessary. The managers supported the project unflaggingly, possibly fearing that the failure of the flagship application would produce domino-like conclusions of failure for the Andrew File System and the Andrew Toolkit, on which the message system was based. The funding had been secured for several years by the initial CMU/IBM contract, so there was essentially no one inclined to put the brakes on the project. Thus, a project that might have appeared to be headed for failure in the early years succeeded in some measure because it was given enough time to evolve naturally. Many other promising projects have surely died due to the absence of such patience and stability.

One useful practice that helped ensure that changes made to Messages would be viewed as *positive* was that the author kept a permanent log of all functional changes made to the system. As the system matured through over one hundred releases of the software, this list became increasingly important. When changes were contemplated, the list could be used to determine why the current functionality worked the way it did. Without this list, it is easy to imagine an endless cycle of changes that undid each other to please diverse audiences. The list made it easier to relate new user feedback to the earlier feedback that had shaped the prior evolution of the system.

It is interesting to note that while the Messages interface grew into a form pleasing to experts and novices alike, it did not do this smoothly or continuously. After the disastrously unpopular first release in the spring of 1986, the next few versions were targeted explicitly at increasing the satisfaction of those who were currently using the system, and thus displayed an increasing bent towards expert users. Later, with the influx of new students in the fall, concern shifted abruptly to the difficulties experienced by new users of the system. This pattern continued for several years -- expert-oriented refinements occurred in the spring and summer, and novice-oriented work was concentrated in the fall and winter. Good user interface projects are often driven by the needs of their users; in this case, the structure of the academic year was a fortunate coincidence that helped keep the Messages interface balanced between novice and expert concerns.

As the system developed, one of the last major pieces to be put in place was the "Set Options" interface. The evolutionary process just described had created a somewhat schizophrenic user base, with an artificially strong division between the novices and experts. Experts would request a new feature, it would be added, and an announcement would tell them explicitly what magic operation they had to perform in order to enable the new feature. But while established experts were able to assimilate one new bit of magic at a time, the growing body of such magic gradually became a major hurdle that prevented new users from growing into experts. That problem was substantially solved with the introduction of "Set Options."

Probably the hardest part of the evolutionary process was determining, whenever an expert-oriented change was made or contemplated, how that change would affect novices, who were rarely part of the discussion about the functional change. It is very difficult for experts to predict how novices will react. Thus it is often hard to determine whether or not a new feature should be available by default. Indeed, the wrong decision was made on more than one occasion, though this was only found out via feedback from later novices. The only useful principle in this regard is to at least make an effort to view each new feature through novice eyes; this will catch many, though not all, of the potential problems. The remainder simply have to be caught by experience with future novices.

To the authors, in hindsight at least, much of this appears to be little more than the application of common sense to practical user interface design. It is worth pausing, therefore, to consider why the myth of the power/usability tradeoff is so widespread. Here, too, the answer is mostly common sense: the above approach to interface evolution is quite costly, frustrating, and time-consuming. It is sufficiently hard and rare to build an interface that is exceptionally good for novices, or exceptionally good for experts, that most projects are more than satisfied with either achievement. For that reason, many users have rarely, if ever, been exposed to an interface that works well for both categories of user. The myth, then, is a simple case of unjustified extrapolation: if I've never seen an elephant, then elephants must not exist.

Unfortunately, the analogy may apply equally to the future prospects for interfaces that work well for novices and experts. Like the elephants, which are being slaughtered wholesale for their ivory, such interfaces may be almost doomed to extinction by the laws of economics. It is far from clear that there is any substantial economic advantage to building programs that are tuned for both novices and experts, but it is all too clear that building them in such a way entails substantial extra costs. It seems sadly unlikely, therefore, that we will see a proliferation of such programs in the near future.

Putting it all Together: Cooperative Work in the Andrew Message System

The Andrew Message System has proven to be exceptionally popular with its user community in general. Weekly statistics indicate that roughly 5300 people use it at Carnegie Mellon to read bulletin boards regularly. Even more users read their personal mail with the system. The AMS is also in use at over a hundred other universities and research sites. This would be indication enough that the system is a success; however, the greatest enthusiasm has in fact been found among those who are using the AMS for substantial cooperative activity. Most notable among these devoted users are the people who provide support services on Andrew at CMU. The Andrew Advisor is a singular example of real-life cooperative work, conducted with the Andrew Message System.[4]

The Advisor System

Centrally supported, distributed UNIX computing at CMU has a long and diverse history. The most recent milestone is the Andrew Project, as described above. Quite apart from the Andrew project is the much longer tradition of departmental UNIX computing, especially among such UNIX sophisticates as are to be found in the School of Computer Science. This tradition is a major influence on the development of centrally supported, distributed UNIX computing. Indeed, "collaboratively supported" is a better phrase than "centrally supported" since it indicates the (sometimes stormy) marriage of departmental and central facilities, systems administration, and user services.

The central computing organizations at Carnegie Mellon face unusual challenges in supporting their computing constituency. Four factors complicate the task. First, the distributed UNIX computing environment we provide has grown substantially beyond the Andrew project, and is now a complex assemblage of vendors' operating systems, the Andrew File System (now provided by Transarc Corporation), the X11 windowing environment from MIT, the Motif user interface offerings from the Open Software Foundation, third-party and campus-contributed software, and, of course, the components

---

[4] Substantially different versions of the following discussion of the Advisor electronic mail consulting service have appeared in Borenstein & Thyberg (1988) and Thyberg (1988).

of the Andrew project: ATK and AMS. Furthermore, this environment is provided and supported on hardware from many manufacturers. Second, although the environment has been widely deployed and promoted, it is an ever developing, rapidly changing environment. As a result, it is not too inaccurate to characterize the computing environment as a 9000-user beta-test site. Third, campus computing expertise is widely, but unevenly, distributed. The users span the entire spectrum from technophobe to technophile. Fourth, the people involved in software development and maintenance, system administration, and user services belong to several organizations and work in different buildings.

To cope with these challenges, members of the Distributed Workstation Services group (DWS), with the help of the AMS group, developed an extensive electronic mail consulting service called "Advisor." Advisor presents to the user a single, private, and personal help resource for every conceivable problem a user might encounter in the complex system described above. The user simply mails a query to Advisor’s account. In 24-48 hours, private mail comes back to the user from Advisor’s account, prepared by a DWS staff member. In fact, however, Advisor is the front-end of a vast network of bulletin boards that enlist the cooperative efforts of all the professional staffs in the central computing organizations.

Advisor I

Advisor has been in use since January, 1985. In the earliest days, it was simply another Andrew account. One person logged in as "advisor," read the incoming mail, handled it with what limited tools were available (online lists, hardcopy lists, hand written notes, and a good memory for the status of a given request), gathered information by talking with the programmers, and sent out replies to the user. This worked reasonably well during the pilot deployment of Andrew when there were a small number of carefully selected users and the Andrew consultant had an office among the Andrew developers.

The first public Andrew workstation lab appeared in the spring of 1986. Shortly thereafter, Andrew accounts were made generally available. Advisor was immediately overwhelmed with mail. An additional consultant picked up Advisor duties, but there were always problems with how to divide the work between the two staff members and how to keep track of the status of any given message. A rudimentary method for classifying messages did exist, but the mechanism was clumsy, time-consuming, and not that useful because all the messages were lumped together in one large, flat mail directory. The combination of the large volume of the easy questions and the genuine difficulty of the hard questions made it difficult to process Advisor mail in a timely fashion. We clearly required some way of getting almost immediate assistance from the right people in the other organizations.

In the fall of 1986, the first version of what is now the Andrew Message System was released to campus. It marked a major advance in the integration of electronic communication. Personal mail and bulletin boards, though conceptually distinct, were now no longer different in kind. A public bulletin board and a user’s private mailbox are both examples of message databases. The only real difference is the degree of accessibility to other users. As indicated above, the AMS supports a rich and flexible set of protection options that permit the creation of public bulletin boards, private bulletin boards, official bulletin boards, semi-private bulletin boards and shared mailboxes, and other variations on the theme. Furthermore, since message databases are built on top of the UNIX hierarchical directory structure, bulletin boards could now be nested within each other.

One of the authors hit on the idea of using bulletin boards as folders for classifying Advisor's mail. The authors created a suite of semi-private bulletin boards, postable by the whole community, but readable only by those in the central computing organizations, and wrote a program in a primitive stack-oriented language for automatically filing messages. (The stack-oriented language was the predecessor to the FLAMES language described earlier.) The result was Advisor II.

Advisor II

Tom Malone, in his discussion of the Information Lens system in Malone, Grant, Turbak, Brobst, & Cohen (1987), has identified three fundamental approaches for handling large volumes of electronic information. The first approach, *cognitive filtering*, attempts to characterize the contents of a message and the information needs of the recipient. The system then matches messages about XYZ with readers who have expressed an interest in XYZ. The second approach, *social filtering*, focuses on the relationships between the sender and the recipient. In addition to the message's topic, the status of the sender plays a role in the reader's interest in the message. The final approach, *economic filtering*, looks at implicit cost-benefit analyses to determine what to do with a piece of electronic mail. Advisor II relied heavily on both cognitive and social filters as the criteria for automatic message classification.

Each message to Advisor that did not come from a member of a known set of Advisor "helpers" was assumed to be from a user requesting assistance. The message was then placed on a bulletin board called "*advisor.open*." The Advisor staff subscribed to this bulletin board and used it as an inbox for new questions. A copy of mail from the user was also placed in *advisor.trail,* to assist the staff in keeping track of requests, and to *advisor.qa*, to which answers would also eventually go, thus forming a repository of useful past work. Thus, the first criterion for sorting the mail was a social one - is the sender a helper or a user? The list of the helpers, that is, the staffs of the various computing organizations, had to be kept current as constants within the stack language program that did the automatic filing of messages.

An incoming question from a user was also copied to one of a series of subject-specific bulletin boards, according to keywords in the subject line. For example, if a subject line was "mail bug," the message was copied to *advisor.mail*. These bulletin boards, though not open to the public, were readable by the developers, system administrators, etc., who subscribed to the bulletin boards covering their areas of interest and responsibility. To continue the example, the AMS group members subscribed to *advisor.mail*, thereby increasing the likelihood of seeing only those messages generally relevant to them. Uninformative or nonexistent subject lines caused the message to be copied to *advisor.misc*. All good Advisor helpers were expected to subscribe to *advisor.misc*, in addition to their other subscriptions. Thus, the second criterion for sorting mail was a cognitive one - is the mail likely to be of interest to a particular group of people?

Cognitive and social filtering were combined at several critical junctures. For example, when the Advisor staff requested more information from the user, Advisor received a blind carbon copy of that request. Because the message was from Advisor, it did not go into *advisor.open* by virtue of the social filter which stipulated that Advisor was never to be taken as a user asking for help. Instead the message went to *advisor.trail* and to the relevant subject-specific bulletin board by virtue of cognitive filtering of the subject line. Another example was in the processing of contributions from Advisor helpers. A helper would see a question on some topical bulletin board. By choosing the "**Reply to Readers**" menu option (which prepends "Re:" to the same subject line as the user's initial post), the helper sent the answer, not to the user, but directly back to that subject-specific bulletin board. By virtue of social filtering, mail from helpers never went into *advisor.open*, but only to some topic-oriented bulletin board. And when a final answer was sent to the user, the blind carbon receipt once again bypassed *advisor.open* because it was from Advisor and ended up on *advisor.trail* and the correct topical bulletin board. In addition, the Advisor would carbon copy the final answers to the *advisor.qa* bulletin board. Unfortunately, the questions and answers were not paired, but in chronological order, due to early limitations in the AMS.

To summarize: the Advisor staff answered questions from *advisor.open* as they were able. They kept an eye on the relevant subject-specific bulletin boards for help with the difficult problems. Having collected the information from the helpers, the Advisors sent polished answers back to the users. As far as the users could see, they had sent mail to Advisor and received an answer from Advisor. The fact that there was additional internal consultation was kept behind the scenes.

Evaluation of Advisor II

The key feature of the first automated Advisor mechanism was the automatic filing of messages into subject-specific bulletin boards. The positive effect of this was two-fold. First, messages came to the immediate attention of the other technical groups. Often, the Advisor staff found that someone in another group had already answered the question before Advisor had even looked at it. This kind of proactive assistance was greatly appreciated. Second, because requests for more information and final answers passed back to the subject-specific bulletin boards, the other groups could provide problem-solving advice and assure technical accuracy.

However, the negative effects outweighed the positive. First, poorly phrased questions from the users led to many "misclassifications." The message filing algorithm worked quite well, but so many subject lines were virtually contentless, e.g. "Help!," that far too many messages ended up on *advisor.misc*: close to fifty percent of all mail to Advisor, according to the authors' estimate. Without better characterization of the message's content in the subject line, the Advisor staff were helpless to get the right mail to the right parties. The designers of Advisor considered the possibility of also searching the body of a message for sort keys, but the pre-FLAMES filtering language was not powerful enough to support free-text information retrieval techniques. Advisor settled for pattern matching on the subject line, rather than suffer too many false keyword hits.

Second, with every question going to a subject-specific bulletin board, the Advisor helpers had no easy way to distinguish between the questions the Advisor staff knew how to answer and those they didn't. Hence, they wasted time answering some questions unnecessarily and neglected other questions for which help really was required. In retrospect, it seems like a truism, but actual use of the mechanism vividly showed that cooperative work disintegrates if what is expected, and from whom, are not clearly articulated. Computer-supported methods can just as easily exacerbate the problem of undefined expectations as alleviate it.

Third, because every blind carbon from Advisor and every message from an Advisor helper also went to the subject-specific bulletin boards, these soon became too cluttered to be of much use. On the one hand, helpers got tired of wading through them. On the other hand, Advisor, at that time, had no way to show a message and all the replies to it in a single chain, so it was sometimes very hard to find the answers that were already available. There is nothing so deadly to cooperation as seeming to ignore another's efforts. Despite Advisor's best intentions, this problem appeared far too often.

Fourth, because every question and every answer went to *advisor.qa*, but the question and the answer were not adjacent messages, *advisor.qa* proved to be virtually worthless as a resource for the Advisor staff.

These four failings were compounded by the rapidly growing amount of mail being sent to Advisor. More staff were needed, contributing to difficulties working from a single inbox, and the helpers were becoming frustrated beyond their willingness to assist in the support of Andrew. It was clear that Advisor needed a significant overhaul.

Advisor III

The third version of the Advisor system was implemented in 1988, and, with the exception of the recent changes described below, Advisor III represents the current state of the system. In Advisor III, the only automatic sorting of incoming mail is by the day it arrived. This sorting is done by a FLAMES program. Mail goes into one of *advisor.inbox.monday*, *.tuesday*, etc. Student Advisors are each responsible for a particular day's worth of Advisor mail. They acknowledge every piece of user mail, handle most of the requests, and then cross-post the tough questions on topic-oriented bulletin boards with names like "*advisor.helpbox.mail*." Figure 6 gives a sampling of the current suite of helpboxes. They are very similar to the "magazines" mentioned previously -- they are, in essence, journals compiled by the Advisor staff of just those questions that require the help of some other group to answer. The technical staffs subscribe to appropriate helpboxes and to the parent bulletin board, *advisor.helpbox*. Posts to the parent bulletin board notify Advisor helpers of the creation of a new helpbox, give a synopsis of its purpose, and invite them to subscribe. All this is done automatically, via folder subscription invitations, one of the "active message" features mentioned above.

Some members of the technical staffs prefer to receive as personal mail the postings to the helpbox they've agreed to monitor. FLAMES makes it trivial to combine any helpbox with a distribution list of interested individuals: these helpers get direct mailings while the bulletin board serves as a shared archive. The helpers' replies go back to Advisor's mailbox, where the FLAMES program processes them and, on the basis of a special reply-to header, places them on the correct helpbox and sends them to any associated distribution lists. The Advisor on whose day the question came in collects the information posted to the helpbox and sends a well-crafted reply to the user.

In addition to the helpboxes, there are *advisor.questions* and *advisor.trail* which provide rudimentary measurement and tracking. Copies of the incoming user mail get placed in *advisor.questions* and *advisor.trail* automatically, thanks to the FLAMES program. Monthly daemons take messages off these bulletin boards and archive them in date-stamped subsidiary bulletin boards, for example, *advisor.questions-Apr-1990*. There is even an Advisor bulletin board, *advisor.daemons*, where the daemons report their activities.

To assist Advisors in getting good answers to the users, a collection of interesting questions and their answers is generated on *advisor.outbox*, which replaces *advisor.qa* from Advisor II. The Advisor uses improved message-filing commands to move back-to-back question/answer pairs to the *advisor.outbox*. Also, there are two bulletin boards for internal dialog; *advisor.discuss*, for meta-Advisor debate and general Advisor information, and *advisor.official* where official pronouncements from other groups can be posted. *Advisor.official* is how Advisor receives such technical and policy "FYI" ("For Your Information") items, insuring that every Advisor sees the information, not just the Advisor on the day the FYI was sent.

It is important to note that Advisor III no longer applies any social filtering to separate the folks who are likely to be qualified to send us official FYIs from those who are not. Staff in other groups who wish to send us an official FYI simply are told to send it directly to the address "*advisor+official*."[5] [6] We apply social pressure on our peers should we ever get information on this channel that is not accurate or useful. In fact, what usually happens is that the Advisors themselves and their supervisors see official pronouncements elsewhere and resend them to *advisor+official*. Another benefit of removing Advisor II's social filtering mechanism is that we no longer discriminate against staff; our peers are able to ask questions of Advisor just as our users do. And by no longer having to maintain lists of who are the helpers, we have been able to expand our assistance base significantly since it is trivial to create and maintain an access group

---

[5] The Andrew Message System interprets any address of the form "userid+text" to be deliverable to the user named on the left of the "+" character. It is up to the FLAMES program processing that user's mail box to take whatever action the user would like, keying off the text to the right of the "+" character. If the user has no FLAMES program, or his FLAMES program doesn't recognize the text, the message is dropped off into the user's mail folder.

[6]

for a particular helpbox using the protection mechanisms mentioned earlier.

Evaluation of Advisor III

By putting human intelligence to work at the heart of the system, the Advisor staff solved, in one stroke, several of the problems that troubled Advisor II. First, Advisor can support a far more fine-grained suite of helpboxes than it could with automatic filing. Poorly phrased subject lines are less of a concern because humans read the mail and digest its contents before passing it to a topical bulletin board. Second, when an Advisor staff member puts a question on a helpbox bulletin board, everyone knows this means that help is genuinely needed. Third, because clutter does not automatically accumulate in the helpboxes, these have become "high-content" bulletin boards that the programmers and administrators feel are worth reading regularly. The payoff for Advisor is a much more reliable information resource. And just in case there are a number of items pending on a given helpbox, the AMS now has a "**Mark Related Messages**" menu option which puts a marker beside all the messages in a given reply-chain. Advisor rarely misses a helper's contribution in the new scheme. Fourth, *advisor.outbox* is a useful repository of previously answered queries because the Advisors themselves decide to post only those question/answer pairs that are likely to be of future use. The questions are now adjacent to their answers with the addition of the message filing command, "**Append to Folder**," which takes a set of marked messages and adds them to the end of a folder, rather than shuffling them into the folder in chronological order.

In summary, though Advisor III lacks the proactive help and the quality assurance that was evident in Advisor II, the Advisor staff is better equipped to handle the load than before. Currently, Advisor receives, on average, 450 new messages per month; 714 messages received is the current single-month record. Note that these are new requests from users; the total number of messages that pass through the Advisor system, including help from Advisor helpers, requests for more information, and replies to users, averages 50 messages per day, or 1500 per month. The student Advisors do an admirable job of performing triage on incoming mail. Full-time DWS staff now function much more as Advisor supervisors, taking areas of technical responsibility, expediting helpbox requests, and insuring that the answers that go out from Advisor are timely and accurate. Messages in Advisor III filter up "manually" through different levels of expertise: the simplest questions are answered by the students, the harder ones are answered by the full-time consultants, and the hardest are tackled by the programmers and administrators themselves. At each level, humans work diligently and efficiently to minimize time-delays inherent in the system. But all parties involved feel that the Advisor scheme focuses and streamlines their efforts.

There were, however, some aspects of Advisor III that cried out for significant improvement. First, there was the problem of correctly routing follow-up mail to the inbox where the initial mail was placed. For example, if the first piece of mail about a particular problem came on Monday and thus was placed in *advisor.inbox.monday*, how would Monday's Advisor continue a dialog with the user on Tuesday, without having all that mail end up in the inbox of the Tuesday Advisor? If the follow-up mail is delivered to the Tuesday Advisor, parallel processing or deadlock can occur as both Tuesday's and

Monday's Advisors try to figure out what's going on.

Second, we had no good way to track requests to Advisor. We would have liked to be able to find out quickly, for any particular piece of mail from a user, when that mail arrived, who on the Advisor staff first handled it, who in some other organization is working on it now, what is the current status of the item, and so on. This was just one aspect of a larger need for good monitoring tools on Advisor. We needed ways to measure the flow of questions, their types, the steps taken to answer them, and the mean time to an answer for a user.

Third, Advisor handles a huge load of routine items like requests for more disk quota. These are matters that rarely require attention from the Advisor staff, save to pass them along to a system administrator and send the users an acknowledgment of receipt. It would have been nice if it took little or no effort to handle such requests.

Fourth, routine filing operations were tedious and error-prone. For example, when closing an interesting exchange with a user, the Advisor had to move mail, one by one, into *advisor.outbox*. The messages that constituted the dialog were likely to be spread around in the inbox and were not necessarily connected by the same subject line. The Advisor would have to rummage around and find all the relevant messages, get them over to *advisor.outbox* in the correct order, and then delete the entire set from the inbox.

How the designers of Advisor have addressed these concerns, and what issues remain for future exploration, is discussed in the remainder of this paper.

Advisor Today

The Advisor III system was sufficiently successful that the basic scheme has been left unaltered. Incoming messages are still classified primarily by the date of receipt, and then filtered upward as necessary through human action, allowing the simplest questions to be responded to by the least-expert Advisors. However, the authors believe that the powerful automatic classification features Messages provided encouraged over-automation in Advisor II and that Advisor III was in large part a reaction against such over-automation. The further development of Advisor has been evolutionary, incremental, and in the direction of adding more automation back into the system. This time, automatic mail handling features have been added in a much more selective, principled, and informed way than was the case in the crude keyword-classification mechanisms of Advisor II. Automation has been added where it could solve specific problems in the Advisor mechanism, rather than attempting to automate the entire process at once.

Structuring Routine Advisor Actions

While the Advisor designers were concerned to solve in a piecemeal fashion particular shortcomings with Advisor III, the authors believe that a pattern of development has been emerging which can be characterized as the application of the language-as-action paradigm (explicated in Winograd & Flores (1988) and Winograd (1988)) to various aspects of the Advisors' actual work practices. This paradigm, along with the Information and Object Lens work of Malone, et al. (Malone, T., Grant, K., Lai, K.-Y. Rao, R. and Rosenblitt, D. (1987), Lai, K.-Y., Malone, T., (1988)), has guided the Advisor staff toward the semi-formalizing of certain linguistic "steps" that Advisor frequently makes in the "language dance" from initial query to final answer.

We mentioned earlier that sorting Advisor mail by day creates the problem of how Monday's Advisor continues a dialog with a user on Tuesday, without getting in the way of the Tuesday advisor. This problem is solved with the Messages customization facilities mentioned earlier. The designers of Advisor have developed a suite of specialized message sending/replying commands on the "***Advisor***" menu card of the "messages-send" window as shown in Figure 7. These commands, which are also bound to keys, insert a special reply-to message header on the outgoing mail. That mail, and all mail in reply to it, get sorted into the correct day's inbox by virtue of that header. So even though the follow-up reply from the user comes in on Tuesday, it still goes to the Monday inbox, where Monday's Advisor is waiting for it. This mechanism is not fool-proof. For example, a user may send in a piece of mail at 11:59pm on Monday and follow it at 12:01am on Tuesday with another piece of mail about the same matter, but with a completely different subject line. Since no reply from Advisor has come to the first message to provide the hook on which to hang subsequent dialog, the two messages are going to end up on different inboxes and the Monday and Tuesday Advisors are going to have to work it out. Still, the special reply-to header works in most cases to route extended mail exchanges correctly.

Notice in Figure 7 that these commands make no mention of any particular day of the week. The day-specific special message header is correctly inserted by virtue of an environment variable, *DAY*, which conditions the behavior of this single set of commands automatically and appropriately. This variable is set for each Advisor in a personal setup file he invokes whenever he logs into the Advisor account. Should this setup mechanism fail and the *DAY* variable be undefined, the sending/replying commands will prompt the Advisor for which day of the week it is that he is now answering. The Advisor can enter the day on the fly and can also set *DAY* for the rest of the session with the "**Change Advisor Day**" menu action. Staff members who work on more than one day's worth of incoming messages can, in a single Advisor session, trivially switch back and forth between, say, their identity as the "Tuesday advisor" and their identity as the "Wednesday advisor." With a single operation, they change all of the special header information that identifies and tracks their correspondence in these roles.

The second problem, tracking the actions that have been taken in response to a user's request for assistance, is one that Advisor continues to wrestle with. To provide the hooks for a solution, the Advisor staff introduced the notion of special message headers that indicate the "state" of each piece of Advisor mail in the progression from initial acknowledgment to closure. State is automatically set by use of the four sending/replying commands shown in Figure 7: "**Acknowledgment**," "**Request for Information**," "**For Your Information**," and "**Final Answer**," each of which marks the outgoing message with a distinct state message header: "`ACK`," "`RFI`," "`FYI`," and "`ANS`," respectively. A reply from the user to an Advisor message of a particular state can inherit the same state message header, which in turn can be processed by either Ness or FLAMES to generate rudimentary tracking and measurement. For example, one could go to advisor.trail, start with a user's initial request, and trace the entire exchange, noting Advisor's acknowledgment of the query, all requests for and provision of further information, and what Advisor believed to be the closing message. If the user replies to that "final answer" it indicates that the matter is still open. Unfortunately, there is currently no way to go back and change the state of Advisor's first "final answer" to something like "first try at an answer," "second try at an answer," and so on. As we have said, tracking a user's request is not yet fully developed in the current Advisor system.

The third area of concern in Advisor III is that of quickly handling the large volume of mail that requires nothing more than "message-shuffling" on Advisor's part. The most frequent request of this sort is the request for more disk quota. The Advisor neither dictates nor applies the quota policy and does not have the privileges required to actually change a user's quota. Thus, the Advisor does little more than acknowledge the user's request and pass it along to the Accounts group, who make the judgment whether additional quota should be granted and perform the necessary steps required to increase the user's quota. To streamline handling quota requests, the Advisor staff created the pair of menu actions "**Quota**" and "**Quota Reply,**" also shown in Figure 7. First the Advisor chooses the "**Forward**" menu action to create a message-sending window with the user's mail in it, giving the Advisor the opportunity to make annotations if warranted. Then the Advisor chooses "**Quota**." The user's message is automatically addressed to *advisor.helpbox.quota*, and a command is run to generate some information about the requester's current disk usage. The results from this command, which are captured in a distinctive font, are prepended to the user's text and the resulting message is sent off with the state message header, "`Quota`," which gives us a hook for measuring the number of quota requests Advisor processes. The message also has a modified reply-to header so that both the user and Advisor will be notified by the Accounts group when the user's quota request has been processed. The Advisor acknowledges the user by using "**Quota Reply**," which sends a message containing a prepared text about policy and current resource constraints.

The pair of commands, "**Helpbox**" and "**Helpbox Reply**," are simply generalizations of the quota operation. After choosing the "**Forward**" menu action, addressing the mail-to-be-forwarded to the correct helpbox, and adding any commentary the Advisor thinks will be useful to the readers of that helpbox, the Advisor chooses the menu action, "**Helpbox**." The state message header "`Helpbox`" is added to the message and the message goes to the specified helpbox. The state message header is a hook both for tracking Advisor's actions in getting an answer for the user, particularly to remind one of pending requests for assistance, and for measuring the frequency with which Advisor asks for help from the technical staffs.

The fourth problem with Advisor III was the clumsiness of certain filing operations that Advisor performed frequently. Compound commands on the "***Classify***" menu card of the messages-reading window, shown in Figure 8, were created to make these actions easy. The menu action "**Current -> Outbox**" appends the currently displayed message to *advisor.outbox* and removes it from the inbox. The menu action "**Related -> Outbox**" gathers the messages that are in the same reply-chain as the currently displayed message, appends them to the outbox, and removes them from the inbox. If necessary, the Advisor can generate a reply-chain with the "**Mark Related Messages**" menu action, mark additional relevant messages by pointing and clicking, and then use the menu action "**Marked -> Outbox**" to move the entire group of messages to the outbox, deleting them from the inbox.[7]

In summary, the four problem areas for Advisor III have been attacked by putting some structure into common Advisor behaviors. The designers of Advisor have made some investigation of the varying illocutionary implications of such linguistic actions as sending an acknowledgment or requesting more information. Though there is much more fruitful development to be done in this area, the authors are satisfied that this kind of approach is the right one for the principled addition of automation to the Advisor service.

Linking Support Groups

The Distributed Workstation Services group has for some time been exporting the Advisor concept and connecting the Advisor system to other help groups on campus. The most mature example to date is a bridge between the *advisor.helpbox.datacomm* bulletin board and a suite of bulletin boards attached to an account, dc0m, belonging to the Network and Communications group. Rather than have these folks subscribe to the Advisor helpbox as a second source of input to their group, the Advisor designers created a "hot link" between the two groups. When Advisor puts mail into its datacomm helpbox, it is automatically resent to dc0m with a special header. When someone in Data Communications replies to that mail, by virtue of that header, it comes back directly to Advisor's helpbox, just where the Advisor expects to find it. There are similar links to other groups who employ Advisor-like systems that we have exported for both academic and administrative use. In this way, DWS hopes to provide these groups with a common front-end to the community -- mail to Advisor -- while allowing them to use whatever internal consulting structures suit them best. It is our belief that a large part of Distributed Workstation Services' role is to enable this kind of distributed support.

Revealing Advisor's Inner Workings to Users

Another subtle but useful change has been in making the hidden structure of the Advisor system more visible to sophisticated users. The Advisor system was heavily oriented from the very beginning to the notion that users would simply send mail to "advisor" and the right thing would happen automatically. That this is ideal for novice users is virtually self-evident. However, it has come to seem desirable to give expert users the ability to direct certain kinds of requests more specifically. (This is an interesting parallel to the

[7] Another evolutionary change in the Advisor system has been the development of customized environments for each of the Advisor staff members. Staff members have developed their own auxilliary subsystems, including additional bulletin boards for their own pending Advisor items, and have elaborately customized compound operations defined as well. The move-to-pending menu actions in Figure 7 are examples of a "personal" extension of the Advisor mechanism which has been adopted by all the Advisors.

general effort, in the Messages interface, to accommodate novices by default but to provide powerful and sophisticated features to those who want them.) Thus, for example, an expert can now send a security-related message to "*advisor+security*," and it will be delivered directly to the Advisor staff members concerned with security issues. In this case, not only is the message delivered more directly, it is also more private -- fewer staff members will see what may be a rather sensitive message.

Advisor's Future

Structuring Routine User Actions

It would be nice if the Advisors did not have to handle such commodity services as quota requests, but had them forwarded immediately to the staff who do take care of such matters. However, the experience with automatic classification by keywords in Advisor II suggests that a simple keyword-based approach to routing such messages might well backfire. Instead, the Advisor staff is developing a combination of mail templates, Ness extension programs, and FLAMES programs that permit users to create semi-structured messages, similar in spirit, though not in detail, to those of the Information Lens system (Malone, et al. (1987)), which can then be reliably routed automatically. For example, a user might type a command such as "more-quota" and be presented with a new mail-sending window, containing an interactive form containing various headers, fields, and relevant information, some of which may be filled in automatically. The data thus generated is then used by Advisor's FLAMES program to send an appropriate acknowledgment automatically and route the mail directly to the right place, rather than have it filter through the normal Advisor mechanism. Once we work out the kinks in a limited domain like quota requesting, the Advisor developers hope to follow this prototype with interactive templates and FLAMES parsings for bug reports, requests for new features, and the like.

Automatic Advisor "Claim Checks" and Social Filtering

We indicated earlier that we have not completely solved the problem of routing all mail from a user about a given problem to a single Advisor's inbox. It has been proposed that the FLAMES file which processes incoming Advisor mail immediately sends back to the user a confirming message which will ask the user to send Advisor any further messages about the matter at hand by replying to this "claim check" message. While such a claim check could be implemented today, the Advisor staff feels it makes more sense to introduce this after we have supplied some Advisor-submission templates, because then the claim check that is returned can be made apropos of the type of query Advisor received. And here is where Advisor may introduce social filtering again. For example, if the submission mechanism can automatically generate information about the status of the sender (e.g. faculty, staff, student, which department, etc.), then the handling of this mail, including the initial claim check, can be sensitive to the different needs of these constituencies and the (possibly) different computing policies that apply to various groups.

Structuring Routine Helper Actions

Those who cooperate with Advisor by reading helpboxes and posting information there do so on a voluntary basis. It would be useful to develop tools for the helpers that semi-formalize their uptake of Advisor's requests for assistance. To design such tools will require careful thought about various illocutionary categories like directives -- Advisor messages that attempt to get the helpbox reader to do something (e.g. answer the forwarded question), and commissives -- helper messages that commit the helper to some action (e.g. fix some bug by a certain date). Furthermore, the helpers need some way to transfer ownership of a commitment, and both Advisor and the helpers need tools to facilitate the negotiation of help commitments, especially if they are subject to change as new information and technical and resource feasibilities warrant. Similarly, if the staffing model for Advisor changes significantly from that of an Advisor taking an entire day's worth of Advisor mail to a queue of requests that all the Advisors draw from, then there may be much greater need for internal mechanisms whereby different Advisor staff members can take up, transfer, and close responsibility for individual user requests. The work of Winograd and Flores, especially as it has begun to appear in software products like The Coordinator, is fundamental to our explorations in this area.

Tracking and Measurement Revisited

An experiment has been conducted with the Advisor system to use our FLAMES program to automatically generate an Informix database of tracking information about all the traffic through the Advisor system. This database, which was a course project for a group of students in the Social and Decision Science department, never went into full-scale use, largely due to a lack of programming resources,. Nevertheless, the idea seems very promising, and also points strongly to the lack of database facilities as an underlying weakness in AMS. The Advisor staff is looking for additional resources to take up this project in earnest. The result will be a system parallel to Advisor's myriad bulletin boards that both the Advisors and their supervisors can use to get status on a particular user request, as well as to generate routine statistical measures and reports.

Question/Answer Service for Users

The *advisor.outbox* is a fairly useful collection for Advisor's own use. But the notion of a database or a hyper-document of commonly asked questions and expert answers that grows in step with Advisor's question-answering is what we are aiming for. Such a tool would be enormously valuable to the Advisors themselves, their helpers, and other computing consultation services around campus. With careful user-interface design and expert system intelligence, it could also be most beneficial to the end user, provided that the information was timely, accurate, and easy to navigate. A recent example of the sort of system we would like to graft onto Advisor is the Answer Garden (Ackerman & Malone (1990)).

Other Engines Behind the AMS Front-end

The infrastructure for the Advisor service was put together using AMS bulletin boards as much because that's the tool we had available as because of any intrinsic virtues of bulletin boards. Exploring other computer-based communication technologies would be a useful exercise. For example, computer conferences are a different breed of animal than bulletin boards. It would be very instructive to re-implement Advisor's helpboxes using an advanced conferencing system, one rich in mechanisms for assigning various roles and passing control of "the floor," in order to see how many of the tools for semi-formalizing Advisor and helper behaviors simply fall out as a consequence of the particular strengths of computer conferencing.

At a more fundamental level, it is clear to the authors that the Advisor service has nearly reached the limits of what current AMS bulletin boards can do as information repositories -- AMS does not provide a general database mechanism, but Advisor often needs one. Then again, without AMS and its powerful kit of features and customization and extension mechanisms, the Advisor staff, who are neither academics nor researchers, but practicing consultants and service providers, would likely never have pursued the vision of computer-supported cooperative consulting to the point where such limitations become apparent. When the son-of-AMS is available, whatever that might be, the designers of Advisor are poised and ready to investigate the avenues of development outlined above.

Conclusions

The Messages interface has been highly successful as a user interface, easily learned and appreciated by novices, easily extended by experts, and powerful enough to support major cooperative work applications. Although one such program cannot be considered proof, it lends support to the notion that power and usability are not fundamentally incompatible. It demonstrates one approach to reconciling power and usability, which entails tailoring all default behavior to novices while providing a simple and graceful mechanism by which experts can extend its power.

The evolution of the Advisor system has taught its designers a great deal about computer-supported cooperative work. Our failed experiments have been the most instructive of all our experiences. But with each incarnation, Advisor feels more and more like an enduring technology for user support in times when central consulting services are lean and everyone looks to some form of distributed consulting to ease the load. We realize that we have only begun to scratch the surface, but we feel we are taking the right steps to exploit the ever-increasing power and sophistication of distributed computing in higher education. The Advisor staff, most of whom are not programmers, have proven able to use the expert-oriented features of the Andrew Message System to develop FLAMES programs, customized compound commands, hot links between support systems, Advisor-templates, and interfaces to alternative engines independently, in large measure, of the AMS developers. It is by virtue of putting these tools in the hands of cooperating workers that the Advisor system continues to be an interesting example of how the AMS supports a large, important, complex, "real-life" cooperative work application.

Acknowledgments

Messages is a part of the Andrew Message System, which was developed by Nathaniel Borenstein, Jonathan Rosenberg, Craig Everhart, Bob Glickstein, Adam Stoller, and Mark Chance, and Mike McInerny. Substantial parts of the Messages user interface reflect the suggestions and experiences of thousands of users, but most especially the suggestions of Dan Boyarski, Chris Haas, Chris Thyberg and Pierette Maniago, who devoted substantial time and effort to studying, deploying, and extending the system. The Andrew Message System was built on top of the rich infrastructure provided by the Andrew File System and the Andrew Toolkit, which are themselves the product of a great deal of work by a great many top-notch software developers. The Andrew Message System and the Andrew Toolkit are part of the Andrew software as distributed on the X11R4 tape from MIT. They are freely available to all interested parties.

Advisor II was conceived by Chris Thyberg and implemented by Pierette Maniago, with help from Nathaniel Borenstein. Advisor III was designed and implemented by Chris Thyberg, Pierette Maniago, and Adam Stoller. Recent Advisor extensions are the work of Chris Thyberg, Wallace Colyer, Judith Jackson, Bob Glickstein, and Michael Riccio. And continued thanks go to our frontline Advisors over the years. They are the real answer-givers and they have been an unfailing source of useful suggestions for the improvement of the Advisor mechanism and of distributed user support in general.

Finally, none of the work described here would have been possible without the encouragement and support of some very enlightened and visionary management at both CMU and IBM. This paper was written with the support of equally enlightened management at Bellcore.

Judy Jackson helped substantially with the description in this paper of the Advisor system. The paper also benefited greatly from the comments of several anonymous reviewers, as well as Terilyn Gillespie, Peter Clitherow, Bob Kraut, and Steve Rohall.

References

Ackerman, M. & Malone, T. (1990). "Answer Garden: A Tool for Growing Organizational Memory," *Proceedings of the Conference on Office Information Systems*, Cambridge, Massachusetts.

Borenstein, N. S., & Gosling, J. (1988). "UNIX Emacs as a Test-bed for User Interface Design," *Proceedings of the ACM SIGGRAPH Symposium on User Interface Software*, Banff, Alberta.

Borenstein, N. S., Everhart, C. F., Rosenberg, J., & Stoller, A. (1988). "A Multi-media Message System for Andrew," *Proceedings of the USENIX 1988 Winter Technical Conference,* Dallas, Texas.

Borenstein, N. S., & Thyberg, C. (1988). "Cooperative Work in the Andrew Message System," *Proceedings of the Conference on Computer-Supported Cooperative Work*, CSCW 88, Portland, Oregon.

Borenstein, N. S., Everhart, C. F., Rosenberg, J., & Stoller, A. (1989). "Architectural Issues in the Andrew Message System," *Message Handling Systems and Distributed Applications*, in E. Stefferud, O-j. Jacobsen, and P. Schicker, eds., North-Holland.

Borenstein, N. S. (1990a). *People Are Perverse: The Human Factor in Software Engineering*, in preparation.

Borenstein, N. S. (1990b). "CMU's Andrew Project: A Report Card," submitted to Communications of the ACM.

Borenstein, N. S. (1990c). *Multimedia Applications Development with the Andrew Toolkit*, Prentice Hall.

Borenstein, N. S. (1990d). "MAGICMAIL: A Secure and Portable Language for Active Messaging," in preparation.

Hansen, W. (1990). "Enhancing documents with embedded programs: how Ness extends insets in the Andrew ToolKit," *Proceedings of IEEE Computer Society 1990 International Conference on Computer Languages*, New Orleans.

Howard, J. (1988). "An Overview of the Andrew File System," *Proceedings of the USENIX 1988 Winter Technical Conference*, Dallas, Texas.

Howard, J., Kazar, M., Menees, S., Nichols, D., Satyanarayanan, M., Sidebotham, R., and West, M. (1988). "Scale and Performance in a Distributed File System," *ACM Transactions on Computer Systems*, Vol. 6, No. 1.

Kazar, M. (1988). "Synchronization and Caching Issues in the Andrew File System," *Proceedings of the USENIX 1988 Winter Technical Conference*, Dallas, Texas.

Kazar, M., & Spector, A. (1989). "Uniting File Systems," *UNIX Review*, March, 1989.

Lai, K.-Y., & Malone, T., (1988). "Object Lens: A 'Spreadsheet' for Cooperative Work," *Proceedings of the Conference on Computer-Supported Cooperative Work*, CSCW 88, Portland, Oregon.

Malone, T., Grant, K., Turbak, F., Brobst, S. & Cohen, M. (1987). "Intelligent Information-Sharing Systems," *Communications of the ACM*, vol 30, no. 3, May, 1987.

Malone, T., Grant, K., Lai, K.-Y. Rao, R. & Rosenblitt, D. (1987). "Semi-structured messages are surprisingly useful for computer-supported coordination." in Greif, I., *Computer Supported Cooperative Work: A Book of Readings*, Morgan Kaufman.

Morris, J., Satyanarayanan, M. Conner, M., Howard, M., Rosenthal, D., & Smith, F. (1986). "Andrew: A Distributed Personal Computing Environment," C*ommunications of the ACM*, Volume 29, Number 3, March, 1986.

Morris, J. (1988). "Make or Take' Decisions in Andrew," *Proceedings of the USENIX 1988 Winter Technical Conference*, Dallas, Texas.

Palay, A., Hansen, W., Kazar, M., Sherman, M., Wadlow, M., Neueundorffer, T., Stern, Z., Bader, M., & Peters, T. (1988). "The Andrew Toolkit: an Overview," *Proceedings of the USENIX 1988 Winter Technical Conference*, Dallas, Texas.

Rosenberg, J., Everhart, C., & Borenstein, N. (1987) "An Overview of the Andrew Message System," *Proceedings of SIGCOMM '87 Workshop, Frontiers in Computer Communications Technology*, Stowe, Vermont.

Scheifler, R., & Gettys, J. (1987). "The X Window System," *ACM Transactions on Graphics*, Vol. 5, No. 2, April, 1987.

Stallman, R. (1981). "EMACS, the Extensible, Customizable Self-documenting Display Editor," *Proceedings ACM SIGPLAN SIGOA Symposium On Text Manipulation*, Portland, Oregon.

Thyberg, C. (1988). "Advisor - An Electronic Mail Consulting Service," *Proceedings ACM SIGUCCS User Services Conference XVI*, Long Beach, California.

Winograd, T. & Flores, F. (1986). *Understanding Computers and Cognition*, Addison-Wesley.

Winograd, T. (1988). "A Language Perspective on the Design of Cooperative Work," in Greif, I., *Computer Supported Cooperative Work: A Book of Readings,* Morgan Kaufman.

Figure 1: The main window of the Messages user interface, as it might look to a new user receiving his first piece of multimedia mail.

Figure 2: The message-sending window.

Figure 3: A mail message in which a raster image is embedded within formatted text.

Figure 4: A "vote" message, inviting the reader to answer a question and have that answer automatically sent back to a specified destination.