CMU's Andrew Project:

A Retrospective

Nathaniel S. Borenstein
Bell Communications Research

For publicaton in *Communications of the ACM*

1 Abstract

The Andrew system is an ambitious and complex software project developed by a joint venture of IBM Corporation and Carnegie Mellon University. The major parts of the mature Andrew system are a high level user interface toolkit, a large-scale distributed file system, and a multimedia mail system. This paper presents a critical assessment of the technical and political successes and failures of the Andrew project.

2 Introduction

The Andrew project [Morris85, Morris88], a joint venture of IBM and Carnegie Mellon University (CMU), was conceived in 1982 as an ambitious attempt to design the university's computing environment of the future, with special emphasis on the needs of students and faculty in an academic setting. Not surprisingly, over time the project strayed somewhat from the original goals, growing still more ambitious in some regards and leaving other areas unattended. In this paper, I will assess, in some detail, the successes and failures of the Andrew project, both technical and political.

This article is necessarily somewhat biased, and such biases should always be made explicit. I am one of the principal designers of the Andrew Message System, one of the three major components of the Andrew project, and the author of a book about the Andrew Toolkit, one of the other major pieces. It might be expected that such a background would bias me towards an unduly positive assessment of the project. In the absence of a truly disinterested but well-qualified observer, however, I undertook to review the project as critically and dispassionately as possible. Inevitably, I have taken positions that are not shared by some of my colleagues on the Andrew project. All opinions should be regarded as my own. The reactions of some of my colleagues who have reviewed this paper have led me to suspect that, if anything, I have been too critical in my assessment, but I suspect that it is more appropriate for me to err in that direction than to be overly positive about a system I played a major role in building.

3 The Origins of Andrew

The Andrew project had its origins in a CMU "Task Force on the Future of Computing", chaired by University Professor Allen Newell. The task force’s unpublished but prescient report correctly identified several of the trends that would come to characterize computing in the 1980’s, and probably beyond. Its major recommendation was the creation of an organization to develop and refine a prototype computing environment for academic use. The idea had great appeal to IBM, which until then had suffered a somewhat checkered history in the academic computing market, and IBM soon agreed to fund a substantial effort to develop such an environment.

The resulting joint venture, the Information Technology Center (ITC), had a broad mandate to design and develop whatever software environment it determined to be necessary for the future university computing environment. The original vision was extremely vague, but was generally perceived as including the creation of educational software. Instead, the project focused, from the beginning, on a somewhat lower layer of tools that were perceived to be prerequisites to properly supporting educational applications. Like many academic development projects, the focus quickly shifted from specific applications to tools to support such applications. It was at this level of underlying tools that the ITC concentrated its efforts, and had its greatest successes. (Indeed, in the early years of the project a large portion of the effort was devoted to such infrastructure as a completely interworked campus network. This effort was successful, but is no longer generally seen as part of the "Andrew" project, even though it was funded and organized as such.)

Although the CMU community was somewhat surprised by the shift away from directly educational applications and towards tools, this was a process of gradually-increasing awareness, at CMU, of what the ITC was doing. IBM’s false expectations regarding the project were shattered much earlier, when the ITC politely rejected IBM’s generous first offer of equipment, a large number of IBM computers that IBM considered, in 1982, to represent the state-of-the-art in the PC world. Instead, the ITC pressed ahead with spending IBM’s money on UNIX-based Sun workstations. Where IBM had perhaps initially seen the ITC as an organization that would develop software to make its computers more appealing in the academic world, it quickly came to see the ITC as more evidence of the need to enter the world of UNIX workstations. Indeed, the development of Andrew and the IBM UNIX workstation product line proceeded in a mutually complementary and beneficial manner.

The ITC itself, on the other hand, developed a fairly coherent idea of what it was building very early in its history. After the first few months, there was a general consensus on the two main parts of Andrew: first, a user interface, originally known as "Virtue," much of which ultimately came to be known as the Andrew Toolkit, or ATK, and, second, the Andrew File System, or AFS, formerly known as "Vice." Although each evolved considerably over time, the two parts have remained the primary pieces of the Andrew system throughout its life. A third part, the Andrew Message System, or AMS, was added in the third year of the project as a "flagship" application to more clearly demonstrate the usefulness of the two main components by providing a multimedia mail and bulletin board system with an extremely large, globally-visible multimedia message database.

3.1 Andrew as an Integrated System

Andrew was conceived and implemented as an integrated system, and as such it is considerably more than the sum of its parts. What Andrew provides to its users is a relatively seamless, integrated computing environment in which the interaction of the various pieces produces a rich set of appealing and powerful capabilities for the end user.

It should first be noted that it is still generally impossible for interested parties to get *all* of the pieces of Andrew from a single source. (Historically, for many years it was impossible to get some parts of Andrew at all, though this is no longer the case.) Most users of Andrew outside CMU, for example, get Andrew from the MIT X11 tape, which does not include the Andrew File System. Thus the maximal version of Andrew, the "integrated system" to be described shortly, exists only at a handful of sites. Andrew users at other sites, though they have some large and useful pieces of the system, often have only a sketchy and vague understanding of the complete Andrew system. That they find even this subset of Andrew to be extremely valuable is in some measure indicative of the even greater strength of Andrew as an integrated system. It is worthwhile, then, to dwell briefly on what Andrew as an integrated system looks like to the end user.

The complete Andrew system provides its users with a strong sense of being a community. The Andrew File System provides a uniformly-visible file system, in which all files are equally visible from any machine at a site (providing, of course, that the user has permission to see the files at all). While this may sound much like the environment provided by other network file systems such as NFS, the uniformity of the environment provided by AFS is important psychologically. Andrew files have truly global names and are automatically visible everywhere, while NFS files do not. At NFS sites, for example, you might be told that a file is in a certain location, and that you have to mount a certain file server first. Even if it turns out that the file server is already mounted, this conditional description of a file’s location is cumbersome and can be intimidating. Casual users -- secretaries, students, and managers, for example -- may not know how to mount an NFS server, or may not have permission to do so. Beyond this, separate NFS administrative domains may not interoperate well with each other, particularly when the domains maintain separte password files and other resources. The net result is that though all the files are accessible in theory, only the real "wizards" are comfortable enough to have confidence that they will be able to find them. Using AFS, a file is specified by a path name, such as "/afs/andrew.cmu.edu/usr/nsb/.signature," and that’s the whole story; anyone who knows how to read a file on UNIX (and who has appropriate permissions) can read the file from any AFS-connected machine, even at another site half way around the world, where the user has no account. The transparency and simplicity of the system empower casual users to cooperate with remote colleagues without learning anything about file servers or file transfer programs. PC and Macintosh users can become a part of this uniform global file system by running software that connects them to an intermediate server. The net result is that people feel connected to each other, the way they felt in the days when everyone at a site had their files on a single timesharing system.

Contributing further to the sense of electronic community is the rich communication environment provided by AMS. By exploiting the vast resources of AFS, an AMS installation can easily provide thousands of bulletin boards for its users to conduct discussions on extremely specialized topics. (Some instructors and students at CMU, for instance, have been known to create a dozen or more bulletin boards, differentiated by topic and by access restrictions, for a single class.) Moreover, the rich editing facilities provided by ATK make the notices that appear on these bulletin boards easier to read and understand than in most other mail and bulletin board systems, by permitting the use of multiple fonts and other formatting features, and by permitting the inclusion of pictures, audio, animations and the like when appropriate. Between personal mail, discussions on the more serious bulletin boards, scanned-in cartoons on the cartoon bulletin boards, and the selected notices that appear on user-edited electronic magazines, people using AMS come to feel a part of a community without ever meeting anyone in person.

Because the ATK multimedia datastream provides a high-level mechanism for applications to interchange data, the various Andrew applications seem to fit together more seamlessly than in many other computer systems. If a user sees an interesting animation on a bulletin board, for example, he can easily cut it out of his mail-reading program and paste it into a file of interesting things he has been saving. There is never any need for format translation to exchange data between applications. Users don’t need to understand this process in order to understand that things simply work together well. (Unlike such seemingly-similar systems as the Macintosh clipboard, ATK data intercahnge is performed at a very high level. Structured drawings or animations, or even user-developed insets or media types, are cut and pasted without being downgraded to bitmaps.)

When Andrew runs on a single workstation, a single user gets the benefit of this kind of integration, but when it is nearly ubiquitous at an entire site, the benefits grow exponentially. When all the pieces of Andrew are put together, what results is an integrated computing environment where even non-technical users begin to feel "at home." A significant portion of the CMU student body, in particular, uses the system every day. In a typical day’s session, a student might read class-related discussion on academic bulletin boards, view an animated plane on the "hobbies.aviation" bulletin board, and store and retrieve mutimedia or textual files shared with users across the campus, without necessarily even knowing that a network file system was in use. Figures 1 through 9 show several reasonably typical examples of Andrew in daily use. More examples, along with additional lessons from the Andrew project, can be found in [Borenstein91a].

4 Evaluating Andrew: The Good, the Bad, and the Ugly

Because Andrew is a large project with several rather different kinds of goals, there are several different kinds of questions that might be asked in order to evaluate Andrew's success:

> **Technical**: Has the Andrew project demonstrated useful technical innovations?
>
> **Implementation**: Has the Andrew software been written well-enough to stand the test of real use?
>
> **Deployment**: Has a significant user community with serious alternatives picked up Andrew and begun to rely on it?
>
> **Goal satisfaction**: Has Andrew achieved anything like what IBM or CMU intended when the project was first begun?

The scope of the Andrew project is such that a proper evaluation of its success will consider all of these factors in the context of each of the major Andrew components. It should be stressed that the evaluative judgments expressed here are my own, and are not necessarily shared by any of the persons or institutions involved.

4.1 The Andrew File System

The Andrew File System [Howard88a, Kazar88, Kazar89] has been, in many respects, the most successful part of Andrew. On a technical level, its successes are clear: when compared to other commonly-used network file systems such as NFS, it has significant performance benefits in all but the smallest installations, and it shows spectacular economies of scale as the installation size increases [Howard88b]. Moreover, it successfully provides a more secure protection environment and a more homogeneous file system than most networked file systems in general use. Most important, AFS has developed into an extremely reliable tool, with remarkably high availability over time. (Comparisons to NFS may be considered unimpressive by those who research advanced file systems, but a major goal of the AFS developers was to produce a robust, practical, and usable file system, which makes NFS the most obvious target for comparison. Other file systems have some, perhaps most, of the features of AFS, but none has apparently been carried through to the level of usability, reliability, and widespread availability of AFS.)

A measure of the success of AFS is the increasing commercial and research interest in the technology. In 1988, a workshop was held to discuss the eventual possibility of a nationwide file system based on AFS, sometimes referred to as NAFS. By 1991, NAFS had become a nascent reality, with AFS cells operating as a nationwide file system connecting machines at over a hundred separate institutions. A new company, Transarc Corporation, was formed in 1989 for the purpose of commercializing and further developing AFS.

Though successful, AFS is not without its problems. The biggest problems historically were political ones; AFS was not freely licensed to the world the way ATK and AMS were, but neither was it sold commercially, so for many years it was difficult for interested sites to get a copy of AFS at all. Once a copy was somehow obtained, sites often found AFS difficult to administer. These problems have been largely alleviated by the formation of Transarc.

Certain technical problems with AFS seem natural consequences of any attempt to produce a UNIX-based network file system. Most notably, the use of AFS seems to make certain programs (sendmail, in particular) become unreliable, because they were not written in the expectation that basic file system operations (e.g. the read(2) or close(2) system calls) could fail, and are not easily modified to cope with such failures. It is hard, however, to imagine a network file system in which these operations could not conceivably fail.[1] Thus it seems inevitable that UNIX processes that require high reliability will need to be rewritten to work well with a networked file system, as indeed the AMS developers found it necessary to write a new mail delivery subsystem. Another example of this phenomenon is the restrictions AFS places on the use of the UNIX "setuid" feature, which allows one program to always run with the privileges of its owner, regardless of who is running it. This mechanism is so inherently insecure in a networked environment that it is mostly unsupported by AFS, which implies that applications that depend on it often require modification to run successfully on AFS.

Other complaints about AFS are more difficult to evaluate. Some critics have said that the UNIX file system is a bad model to begin with, so that AFS is at best a good imitation of a bad idea. At the same time, others have criticized AFS for its few deviations from

---

[1] An AFS option (on a per-workstation basis) offered the guarantee that close calls would never return until they succeeded, even if that meant several days or weeks of waiting. It would be difficult to claim that this resulted in a system that was actually better for the users, although many NFS sites do seem to prefer these semantics. On such a system, it is worth noting, file operations may be guaranteed not to fail, but this is not the same as saying they are guaranteed to succeed -- instead, they may simply wait forever, not always ideal behavior in, for example, a mail transport agent with a choice of mail queues on several different file servers.

standard UNIX semantics, notably in the protection mechanisms, which feature an enriched notion of access lists, but only on a per-directory rather than a per-file basis. The fact that AFS clients and servers cannot run on the same machine can lead to severely degraded performance in such "pathological" cases as database servers, which benefit enormously if all write operations can be performed on the local disk.

Because AFS was based so heavily on the notion of whole-file transfer, early versions of AFS exhibited extreme performance degradation in working with very large files, which led to a general perception that AFS was unsuitable for database operations. Of course, this ignored the fact that databases do not need to be implemented as single large files, because in practice most databases do indeed use very large files. In contrast, the AMS White Pages technology is implemented using a database that builds a B-tree of small files, and this performs extremely well on AFS. Moreover, more recent versions of AFS have implemented a limited notion of partial-file transfer, in which large files are broken up into smaller chunks for significant performance improvements with conventional database systems.

Finally, there are always complaints that AFS is too slow. No number of studies that purport to show that AFS is faster than competing file systems will suffice to pacify impatient users who have to wait for files fetched from AFS. Personal and anecdotal experience seems to suggest that this will always be a complaint about any network file system. Within days after a major improvement to the speed of the system, the users always seem to get used to the new level of performance and start asking for more. Moreover, there may be a natural tendency, on the part of system administrators, to use AFS efficiency not to improve user-visible performance but to reduce site-wide costs, concentrating a large number of users on a remarkably small number of file servers. This strategy, of course, tends to hide the efficiency of AFS from the users who, in the extreme case, end up with the lowest level of service that the administration can get away with.

4.2 The Andrew Toolkit

The basic architecture of the Andrew Toolkit [Palay88, Sherman90, Neuendorffer90, Sherman91], and specifically the architecture of "insets," user interface objects that cooperate at an extremely high level and share resources in all ATK applications, is probably the most important innovation of the Andrew project. In this regard, at least, the ATK effort must be regarded as a significant technical success.

Other technical successes of ATK are largely related to or made possible by the inset architecture. The notion of multimedia documents is rendered much clearer and simpler by insets. The ATK datastream, a powerful abstraction in its own right, would probably be much less powerful and harder to implement without the underlying notion of insets. The use of dynamic loading in ATK was extremely successful long before dynamic loading became widely available as an operating system-level functionality. Although dynamic loading is largely independent of insets, the inset paradigm made it much easier to determine the module boundaries at which dynamic loading should occur.

From a user’s perspective, ATK is successful in that it promotes a more homogeneous applications environment without sacrificing richness and power. Even the most complex of multimedia objects can be easily cut and pasted between applications, another consequence of the inset architecture. Because insets, rather than applications, are the focus of ATK programming, most functionality implemented for one ATK application becomes immediately available in all the others.

From a developer’s perspective, one of the best things about ATK is the object-oriented environment in which it is written (the "class" preprocessor to the C language). Combining the inherent low-level power of C with the conceptual power of programming by subclassing, and then adding in the building blocks provided by the library of insets, ATK gives programmers an extremely livable compromise between pure object-oriented languages such as Smalltalk-80 [Goldberg83] and traditional system programming languages such as C. The relative ease with which the Ness extension language [Hansen90] and ADEW development environment [Neuendorffer91] could be built on top of ATK is clear testimony to the power ATK offers developers.

Still, there are many negative aspects to ATK. If ATK can fairly be called the most innovative component of Andrew, so too it might well be called the least well-engineered or polished. An easy criticism of ATK is that it is too big, and indeed the libraries are enormous. Much of its size is necessary, for it does a great deal, but some of the size is undoubtedly reflective of overly convoluted and complex and frequently-altered code, and of a proliferation of insets of varying importance and quality.

ATK evolved over several years, as its designers groped towards a reasonable model for supporting user interface applications. ATK’s predecessor, the Base Editor [Gosling84a], had nothing like the notion of insets, and in fact resembled conventional toolkits such as Xt [Swick88] more than they resembled the current Andrew Toolkit. Most of the system has been rewritten along the way, but some of the code in ATK has been carried forward, with modifications, from those ancient versions to the present, leading to historically baroque and inscrutable code. The history and complexity of the toolkit are certainly directly responsible for the complexity of customizing ATK applications; there are at least a half dozen different customization mechanisms for ATK, and their power is significantly undercut by the difficulty of figuring them all out.

A particularly egregious failure of ATK is one of specification: there is nowhere a complete and coherent specification of the inset architecture. The basic notions of what insets are and what they do are explained in numerous places, but a rigorous specification of the definition of an inset can be gleaned only from reading the code. This is a particularly distressing failure, since the innovative nature of insets has led to widespread interest in their precise definition.

The ATK datastream is similarly unspecified. Long experience has taught that a data interchange format is among those abstractions most in need of precise specification, but the ATK datastream was never so specified. Part of the problem is that each inset implements its own datastream. While this is no excuse to avoid the effort of format specification, it does make the problem more distributed: the author of the text inset must write the specification for the text datastream, and so on. Not only was this never done, there was never even a clear specification of the conventions by which the individual insets' datastreams are grouped together to form a larger document, nor of the basic rules which all insets' datastreams must obey. (The only attempt to do so to date can be found in [Borenstein90].)

In addition to being poorly specified, ATK also suffers from certain failures of integration. The ADEW interface building tools, for example, though extremely powerful and useful, would be even more useful and usable if they were more completely integrated into the basic toolkit, rather than built on top of it and outside its core.

Ness, the ATK extension language, stands as an interesting example of what programming languages can become in a world with integrated and cooperating graphical objects. Ness "strings," for example, can include not only multiple font objects, but embedded pictures, music, or any other Andrew insets. Moreover, editing these strings in the Ness program is precisely identical to editing them in any other Andrew document. The power and elegance of these general ATK mechanisms is nowhere made clearer than by their use in Ness. On the other hand, Ness clearly suffers from the problem of security; when programs can be embedded in documents or sent through the mail, security becomes a serious concern. That Ness has any mechanism at all to deal with the security problems of embedded computation in documents is itself novel. However, these mechanisms are, in some measure, inadequate. They require a user-level decision about whether or not to trust and run a program, and it is highly unlikely that most users will be qualified to make the correct decision in context. Ongoing research [Borenstein92] is aimed at improving the security of documents extended with Ness-like programs.

Although the class preprocessor and dynamic loader have provided a substantial amount of ATK's power for the programmer, they have not done so without a cost. The fact that class is a non-standard language is a significant deterrent to many who would consider using ATK, even though it is in fact a small language and most ATK code appears to be standard C. (It is commonly asked why Andrew did not use, for example, C++. The answers are that C++ lacks several features the ATK designers considered crucial, such as dynamic loading and module initialization, and also that it was new, unproven, and not obviously destined for success when the project was begun.) Worse still, the dynamic loader itself is a portability headache. Most of Andrew is readily portable to any UNIX-based machine, but the port of ATK's dynamic loader is more problematic. At the heart of the dynamic loader is the need to deal with the different object file formats on each operating system, and a few dozen lines of assembly code that must be rewritten for each new processor, and sometimes for each new release of an operating system. Historically, this meant that as innovative new processors such as SPARC and MIPS are introduced, with new versions of UNIX to match, their users found that they either had to port Andrew's dynamic loader themselves or wait until someone else ported, tested, and redistributed the relevant pieces of code. Predictably, in several circumstances this helped deter prospective Andrew users from programming with ATK. (This situation has improved as standard operating system interfaces to dynamic loading have become available and have been incorporated into ATK's dynamic loading mechanisms.) On balance, the use of class and dynamic loading seems to have been worthwhile, but the extra power and convenience have been purchased at the price of a new language and a small chunk of unportable code.

A more specific but still fundamental failing of ATK is summed up in the chronic lament of long-time Andrew users: "Why can't Andrew print?" The truth is that Andrew *can* print -- complex documents including embedded insets print routinely -- but that the mechanisms by which it prints are so baroque and fragile as to be more bug-prone than all the rest of Andrew combined. There are a number of reasons for this. First, Andrew suffers from a split personality with regard to printing. Early in the project, the designers rejected the "WYSIWYG" ("What You See Is What You Get") paradigm, under which the screen reflects precisely what will be printed on paper, in favor a philosophy they dubbed "WYSLRN" ("What You See Looks Real Neat" -- a real Pittsburghism), under which the interface is optimized for the current display and does the best it can when printing. This philosophy is more or less inevitable if one takes seriously the notion of embedding active objects such as animations, sound, and video within documents. This is because such objects can simply never be printed, and it would be pointless to make their screen images mirror exactly their inadequate paper representation. But as Andrew marched down the WYSLRN road, it quickly became apparent that whatever magic was being implemented for the screen, the printer was lagging behind. Aggravating the situation was the fact that early versions of Andrew were developed in a troff-based environment, and later versions, producing output for PostScript printers, do so by translating troff output into PostScript. The result is a rather long and convoluted pipeline for printing, in which insets produce troff source code which is piped through eqn (and, originally, through tbl and pic as well) and then to a dvi-to-PostScript translator before being printed. Finding and fixing bugs in such a system is a formidable task. Though still imperfect, printing has become significantly more reliable in recent releases, and complaints have decreased dramatically. Also, if Andrew is more famous than other toolkits for its printing problems, this is in large measure because most toolkits offer no support at all for printing complex multimedia documents!

On a non-technical level, the biggest fault of the ATK project was probably the violent, wrenching changes that often followed each new insight on the part of the designers. On at least three separate major occasions, and innumerable minor ones, changes to ATK required substantial or total rewriting of ATK application programs. This series of "cataclysms" may have helped make ATK much better in the end, but it alienated many ATK users and potential users at CMU and elsewhere. (In particular, the developers of cT, a language and environment for educational software development, parted company with ATK rather than follow it through one of the most traumatic of these transitions.) Although ATK has been more stable in recent years, old-time ATK developers still get nervous when they hear ATK's authors speculate on the "sweeping" changes that might be required, for example, to move to an all-PostScript printing architecture. Many developers would be happier if printing remained less than ideal, but ATK remained stable.

4.3 The Andrew Message System

As the "flagship" Andrew application, AMS [Rosenberg87, Borenstein88] enjoys a great deal of visibility and popularity with the Andrew user community. Disgruntled undergraduates at CMU have been heard to mutter that AMS is "the only decent part of Andrew," and this reputation has even led some computer companies to make inquiries about separating AMS from the rest of Andrew by porting it to their own user interface environment. While all of this is flattering to the AMS developers, it also largely misses the point. A large portion of the benefits that users perceive as coming from AMS are in fact coming more or less directly from the multimedia capabilities of ATK or from the mass-storage capabilities of AFS, and the relatively higher level of user enthusiasm for

AMS must be understood to reflect the fact that AMS is a more visible commodity to the users, who may not even know what ATK or AFS *are*. Moreover, AMS remains the only really large non-proprietary application that has devoted a massive effort to realizing the potential inherent in the underlying ATK and AFS technologies. (Several companies have based large internal development projects on ATK, but the resultant software is not available outside the companies involved.)

There are, of course, some real achievements that properly *should* be credited to the AMS developers. Prior to the AMS project, it was widely, though inaccurately, believed that multimedia mail could not be implemented using the RFC822/SMTP protocols, and would have to wait for X.400 [Schicker89]. Though prior research had demonstrated that this was not the case, AMS reinforced this fact and made more people aware of it, notably including Steven Jobs prior to the design of the NeXT mail system. Compatibility with a heterogeneous mail world was a major focus of the AMS project, so that today AMS-formatted messages pass successfully through all known mail gateways.

The notions of interactive messages and user-edited electronic magazines, though not entirely original, have been extended and popularized by AMS, and have been widely appreciated by users and seem likely to be imitated by future systems.

The white pages user name database has proven extremely popular, and its ability to perform fuzzy name matching based on phonetics and other heuristics provides one of the clearest possible examples of how conventional electronic mail technology can be enhanced to make life easier for the user. The underlying btree database is an achievement in its own right, and demonstrates that a file system based on whole-file transfer need not be permanently deficient in database capabilities.

The message delivery system, AMDS, is an interesting and unique example of a distributed mail delivery agent. The developers paid a great deal of attention to the possibility of partial file system outages, and the ensuing architecture that exploits multiple forms of redundancy and distributed queueing is the very model of an efficient distributed service. Indeed, the AMDS architecture provides so many levels of backup that one can begin to see that a distributed system such as this one might actually be *more* efficient and reliable, overall, than traditional architectures.

Messages, the multimedia AMS user interface, has proven particularly popular with users as an example of a tool that combines ease of use with flexible and powerful options. It is often the first program a new Andrew or UNIX user learns to use and feel comfortable with, and yet it satisfies the needs of demanding expert users as well. [Borenstein91c]

There are several things to be said on the negative side, however. Perhaps the most common criticisms of AMS are that it is too big and too slow. Part of this is due to the fact that AMS is at the top of the Andrew pyramid: by the time you run Messages, the multimedia mail-reading interface, you are running a large application program that includes all of the AFS, ATK, and X11 libraries in addition to the AMS code itself. The result is a program that, when running, often grows to occupy 2 or 3 megabytes of memory, and sometimes more. Much of the code size is attributable to excessive layering and extreme paranoia on the part of the designers, who strove to catch and handle properly every conceivable error situation. (This attempt may seem somewhat futile, and costly in terms of size and speed, when one considers that the ATK libraries, on which the Messages program depends, share little of this paranoia and will die horribly if, for example, the process runs out of memory.) Much of the multi-layered structure of the AMS code was dictated by the ambitious system architecture, which supports remote connections, from client user interface programs running on PC's and Macintoshes, to the UNIX-based "messageserver" process. If, for example, the designers had from the outset intended to build a UNIX-only mail system, they would have been able to dispense with a great deal of complexity and layering (including the whole remote procedure call level) with a substantial reduction in the size and complexity of the code.

Not surprisingly, the developers have observed that the frequency of complaints about the code size is inversely correlated with the power of a user's machine. Moreover, the user often fails to appreciate just how much AMS is doing when it executes a program written in FLAMES ("Filtering Language for the Andrew MEssage System", the AMS LISP-based filtering and message routing extension language) to determine how to automatically classify incoming mail, or when it takes thirty seconds to determine on which, of the two hundred or more bulletin boards Andrew users commonly subscribe to, there are new messages to be read. Nonetheless, having said all of this, it is undeniable that the AMS code is itself considerably bigger than it ought to be, and slower as well.

Another problem with AMS is that SNAP, the AMS remote procedure call package, is fundamentally underpowered, a questionable design decision that has had detrimental effects in several parts of the AMS code. SNAP was designed to be extremely simple, so that it could be easily ported to various low-end architectures such as MS-DOS machines. The price paid for this simplicity includes unidirectionality; the client can call the server and get an answer, but not vice versa, which has complicated the handling of exceptions and events by the messageserver. Another limitation of SNAP is that it restricts the remote procedure call parameters to a fixed maximum buffer size. Although this limit is large, the fact that any such limit exists forced several messageserver calls to transmit data indirectly through temporary files rather than through parameters, a mechanism which increased the complexity of certain parts of the code. In retrospect, the job of porting a more sophisticated RPC package to the low-end machines might have been simpler than the job of coding around such a simple-minded package, although this is still a point of some dispute among the AMS developers.

The AMS database format is one of weaker points of the system, as well. AMS stores messages in an *ad hoc* special-purpose database. Like many such databases, the AMS database is weak on search and indexed retrieval functions, such as searching for all messages with the word "yogurt" in them. In addition, the database format is idiosyncratic -- users cannot easily switch back and forth between AMS and other popular programs such as MH [Rose86], Elm [Taylor87], or standard netnews-reading software. If an underpowered database had to be used, one can easily argue, it should at least have been a compatible one. For efficiency, the AMS database includes a bewildering array of index files; in general, users are happily unaware of the existence of these indices, but system administrators are sometimes dismayed at having to deal with them.

Just as AMS is often praised for features inherited from AFS and ATK, so too it is occasionally criticized for inherited or related shortcomings. Like AFS, AMS can be hard to administer at a large site. Like ATK, AMS suffers from a problem of fragmented and confusing customization mechanisms. In both of these cases, AMS seems to have inherited a bad situation and made it even worse.

Like AFS, AMS suffered for many years from difficulty in making its source code widely available. While most of AMS has always been freely distributed with ATK as part of the X11 distribution, a few separable components (the delivery system (AMDS), the White Pages, and SNAP) were, until 1989, withheld from that distribution, but were also not available as commercial products. Needlessly to say, this was frustrating for those sites that wanted them, who typically had to make special arrangements with IBM and CMU. These problems are no more. All of the parts of Andrew that are not Transarc products are now freely distributed on the X11 tape.

More basic criticisms of the AMS project might be that it has been too far in advance of standards for a development project, and too concerned with real users for a research project. By positioning the project in this way, the developers were able to provide multimedia mail service years before multimedia mail standards became widely available, but opened themselves up to possible criticism for doing work that will have to be reimplemented, perhaps under X.400. The developers' position, on the other hand, is that the primary goal was to provide services to a user community in a timely fashion, that care was taken to ensure compatibility with standards as they evolved, and that much of AMS could actually be ported into a standards-based environment as the relevant standards emerge. It is hard to see how a project can try to be both good research and useful development and avoid such problems.

AMS, ATK, and similar projects such as BBN Diamond/Slate and NeXT mail have added urgency to a problem that is likely to challenge the electronic mail world for many years to come, the problem of the definition of a data format for the interchange of multimedia documents. This is a problem that many people would like to consider solved with the advent of "standard" interchange formats like ODA [Rosenberg90]. However, the Andrew project's work with the ODA Toolkit has proved enlightening in this regard. The strategy taken by ODA, and indeed by all other standards for formatted document interchange, is to rigorously specify all aspects of the appearance of a document. ODA thus includes, for example, excruciating detail about how to specify the font and positioning of text, and the composition and location of raster images. However, such languages are by their very nature forever limited to the set of objects that have been rigorously defined, while systems like Andrew encourage developers to innovate and develop new types of objects frequently. In the ODA model, each newly-developed inset would require a new extension to ODA, an unlikely occurrence. In contrast, a group of mail experts on the Internet, among whom Andrew users including the author have been prominent, have recently produced a draft standard [Borenstein91b] specifying a standard format for multimedia mail on the Internet. This draft, which is highly flexible and extensible in the spirit of ATK, is strongly influenced by AMS, which is itself being upgraded to comply with the new standard.

In advance of a standard format for document interchange, of course, heterogeneous multimedia mail systems are unable to interchange mail, but that isn't even the worst problem. Even multiple Andrew sites will be unable to successfully read each others' multimedia mail if the mail includes objects that have been developed at the sending site but have not yet been imported to the receiving site. The only way out of this dilemma would be to write insets not as dynamically loadable C code, but as interpreted code in some as-yet-unspecified language. Indeed, Steven Jobs has, in conversation with the Andrew developers, advocated using Display PostScript in this role. Aside from the arguable appropriateness of PostScript for this application, this ignores a crucial problem raised by sending code in any language through the mail: such a scheme introduces security problems and Trojan horses on a scale that was previously unimaginable. This is the problem that Ness security features attempt to address, although the Ness solution is clearly not the last word on secure languages. Indeed, the design of an appropriate language for document exchange is clearly a fertile area for futrher research.

A more detailed evaluation of the successes and failures of the AMS architecture can be found in [Borenstein89].

4.4 Other Andrew Applications

The few non-AMS Andrew applications that were developed as part of the Andrew project -- notably the Console system monitor, Typescript shell interface, Ez multimedia document editor, and Help programs -- have been well-received, and are popular with the Andrew user community. The great failure of the Andrew application development effort, however, is that it barely exists. With all the concentration on basic tools, the Andrew developers were pleased to have developed one major application (AMS), a few minor ones, and a core of useful insets (drawing insets, animation insets, and the like). One of the most persistent criticisms that has been leveled at the Andrew developers by the CMU community is that in their obsession with basic tools, they have overlooked the user community's basic need for more and better-polished application programs.

Overall, it must simply be said that the Andrew project chose, for better or worse, to devote its resources to underlying tools with long-term prospects for helping produce better applications for end users, but (aside from the AMS effort) largely declined to work on such applications directly. The hope was always that the AMS effort would make the value of the tools evident, and that others would then use the tools to produce a richer applications base. Although that has not yet happened, it still appears a defensible and reasonable strategy.

4.5 Andrew and CMU

It must be said that Carnegie Mellon has not gotten at all what it was looking for when the Andrew project was first conceived. Although Andrew is widely used on the CMU campus, it is by no means the only computing environment on the campus. Cheaper machines such as PC's and Macintoshes substantially outnumber the more expensive Andrew workstations, and timesharing systems running VMS or UNIX retain a solid cadre of users. More important, the Andrew environment is, from the CMU perspective, woefully incomplete. Although Andrew is popular for its mail and bulletin board system and a few other applications, it is generally outgunned, as far as other applications (spreadsheets, drawing editors, and so on) are concerned, by commercially-produced software on the low-end machines. For campus users, Andrew would be more unambiguously successful if, for example, it ran on a Macintosh and exported the power of ATK into commercial applications. (That this latter idea is not really technically feasible does little to dispel users' belief that this is what they want.) In addition, many application developers (or potential developers) on the campus were "burned" by the historical instability of ATK, in particular, and the tendency of the ATK designers to radically rework the programmer's interface to the toolkit. But the most conspicuous failure of the Andrew project, from the CMU perspective, is that it made essentially no effort to provide real educational applications to the campus, or even to support the development of such applications any more than it supports any other kind of complex user interface. (In contrast, MIT's project Athena put far less effort into developing general software tools, but arguably did a much better job supporting education and the campus. Whether one considers Athena or Andrew to be a greater success depends, in large part, on what one thinks the goals of such projects ought to be.)

Overall, despite these glaring failures, Andrew remains reasonably popular at CMU, where its blend of usability and power is a good match for a community that includes a large number of hackers and neophytes alike. The message system, in particular, is widely used and serves to increase a sense of community on the campus. And, of course, the successes of Andrew in the wider world have reflected well on CMU, a fact that is not lost on the campus community. Neither is the fact that the development of the system didn't cost CMU much money, and served to bring in money, machines, and people from IBM.

4.6 Andrew and IBM

Insofar as one can generalize about such a large and multifaceted company, IBM appears to have been pleased with the results of the Andrew project, although the developers have often wondered why. For various reasons, many of them technically justifiable, the

Andrew project has not been quite the showcase for IBM hardware that the original IBM contract envisioned. Andrew was prototyped, for example, on Sun workstations rather than IBM equipment. Even after the IBM RT PC became the most common Andrew workstation at CMU, Andrew for years was not running on any of IBM's standard operating systems (AIX, OS/2, VM/CMS, or even MS-DOS). Instead, it was running on a version of Berkeley Unix that IBM supported only for university use. Thus even when IBM managed to get potential customers excited about Andrew, it couldn't really begin to sell it to them. (This situation finally ended, in 1989, with the port of Andrew to AIX.) More generally, IBM and CMU had a good deal of trouble working out procedures for technology transfer, and those who wished for a strategic (mainline) IBM product based on Andrew have endured years of frustration.

But if Andrew has not, from IBM's perspective, been a direct money-maker, still the project has not been without benefits. Andrew has probably helped to enhance IBM's reputation within the university, research, and UNIX communities. Andrew also has provided a few minor IBM products at the same university-oriented level as the IBM version of Berkeley UNIX, and (in modified form) some specialized mainline products. Andrew is widely used at several internal IBM sites, where users offer testimony to its value as a prouctivity tool for software development. The bottom line, though, is that whether or not Andrew was successful from the IBM perspective (and worth the millions of dollars spent) depends in large part on whether IBM views it as research or as advanced development. It is clearly more successful as the former than the latter, but it has never been clear (to IBM *or* CMU) which of these is the proper perspective.

4.7 Andrew and the World

The same duality that affects Andrew’s relationship with IBM can be found in the way it relates to the larger world. Andrew is often cited for its innovations, and is probably in daily use at at least two hundred sites. (The actual number is remarkably hard to estimate, since the software is distributed for free on the MIT X11 tape.) While this is successful by some estimations, some of the designers were shooting rather higher, and are predictably disappointed at the slow pace of Andrew’s spread. Viewed positively, the number of sites using Andrew might seem fairly high considering that the software is essentially unsupported -- universities simply can’t provide the same kind of user support one expects from IBM or Microsoft. That most Andrew users have taken this enormous set of software "as is" from the X11 tape, and have found it both usable and useful, is no small achievement.

It is worth noting, however, that, when drafts of this paper were circulated among the Andrew system designers, the previous paragraph probably stirred more controversy than the rest of the paper combined. Some regard the fact that Andrew has not "taken over the world" as evidence of its failure, while others regard "daily use at two hundred sites" as evidence of practical success beyond their wildest dreams at the project’s outset. It seems clear, at a minimum, that there was never a real consensus about the practical goals of the Andrew project, and the lack of such a consensus is now reflected in a lack of consensus about its practical success or failure.

Another measure of Andrew’s success comes in its designation as a "research technology" by the Open Software Foundation, and by the fact that key designers from such companies as Sun and Next have studied parts of Andrew closely, and in some cases have wholly or partially based products on it.

On the other hand, the history of Andrew in the larger world is largely a history of missed opportunities. The old Andrew window manager, WM [Gosling84b], which predated the X window system [Scheifler87], might well have evolved into the standard that X11 is today had IBM made the early versions of Andrew freely available to those who wanted it at MIT. In hindsight, the decision to base the "academic computing environment of the future" on UNIX might also seem questionable to many. Between the dependence on UNIX and the lack of vendor support, it is little surprise that Andrew is virtually non-existent outside the academic and research communities. If the developers miscalculated, it was, in Jim Morris' words, because "we expected big UNIX boxes to get cheaper faster than they did, and we didn't expect little DOS boxes to get smarter as fast they did."

In the final analysis, however, Andrew's greatest failing is probably inherent in the way it was built: because the project quickly diverged from its original goals and never explicitly re-stated its goals as they evolved, it eventually became a nearly impossible task to try to determine whether or not Andrew was succeeding. To this very day, debates rage among the major developers of the Andrew software regarding what it was they were trying to do and whether or not they succeeded. Those who believe that the goal was to build innovative and influential research prototypes generally consider the effort successful. Others, who believe that the goal was to reshape the future of computing and education at Carnegie Mellon, are slightly less satisfied, but are happy overall. On the other hand, some will not consider Andrew a success unless it is being run on nearly every UNIX workstation in the world. If the goals of the project had been more clearly defined and kept up to date, it would be considerably easier to determine, at this late date, whether or not the project was successful.

5 Acknowledgments

The Andrew project is the result of the creative efforts and talents of an enormous number of people, too many to name here. (The editor suggested an exhaustive list, but an initial list of acknowledgements ran to 103 names and was probably still far from complete!) Special credit must be given, however, to Jim Morris, the primary visionary behind Andrew as it actually developed.

An earlier version of this paper was reviewed critically and insightfully by Tom Neuendorffer, who provided many valuable suggestions. Later versions benefited substantially from the comments of Al Buzzard, Craig Everhart, Fred Hansen, John Howard, Bob Kraut, Andy Palay, Jonathan Rosenberg, Mark Sherman, and Susan Straub.

6 References

[Borenstein88] Borenstein, et al., "A Multi-media Message System for Andrew", *Proceedings of the USENIX Technical Conference*, Dallas, February, 1988.

[Borenstein89] Borenstein, et al., "Architectural Issues in the Andrew Message System", *Message Handling Systems and Distributed Applications*, E. Stefferud, O-j. Jacobsen, and P. Schicker, eds., North-Holland, 1989.

[Borenstein90] Borenstein, Nathaniel S., *Multimedia Applications Development with the Andrew Toolkit*, Prentice Hall, 1990.

[Borenstein91a] Borenstein, Nathaniel S., *Programming As If People Mattered: Friendly Programs, Software Engineering, and Other Noble Delusions*, Princeton University Press, 1991.

[Borenstein91b] Borenstein, Nathaniel S., and Ned Freed, "Mechanisms for Specifying and Describing the Format of Internet Message Bodies", Internet Draft (soon to be an RFC), October, 1991.

[Borenstein91c] Borenstein, Nathaniel S., and Chris Thyberg, "Power, Ease of Use, and Cooperative Work in a Practical Multimedia Message System", *International Journal of Man-Machine Studies*, Vol 34, April, 1991.

[Borenstein92] Borenstein, Nathaniel S., "Computational Mail as Network Infrastructure for Computer-Supported Cooperative Work", submitted to IFIP WG 6.5 Conference on Message Handling Systems, May, 1992.

[Goldberg83] Goldberg, Adele, *Smalltalk-80: The Language and its Implementation*, Addison-Wesley, 1983.

[Gosling84a] Gosling, James, and David S. H. Rosenthal, "The User Interface Toolkit", *Proceedings of PROTEXT 1 Conference*, 1984.

[Gosling84b] Gosling, James, and David S. H. Rosenthal, "A Network WIndow Manager", *Proceedings of the 1984 Uniforum Conference*, January, 1984.

[Hansen90] Hansen, Wilfred J., "Enhancing documents with embedded programs: How Ness extends insets in the Andrew ToolKit," *Proceedings of IEEE Computer Society 1990 International Conference on Computer Languages*, March, 1990, New Orleans.

[Howard88a] Howard, et al., "An Overview of the Andrew File System", *Proceedings of the USENIX Technical Conference*, Dallas, February, 1988.

[Howard88b] Howard, John, et al., "Scale and Performance in a Distributed File System", *ACM Transactions on Computer Systems*, Vol. 6, No. 1, February, 1988.

[Kazar88] Kazar, Michael Leon, "Synchronization and Caching Issues in the Andrew File System", *Proceedings of the USENIX Technical Conference*, Dallas, February, 1988.

[Kazar89] Kazar, Michael, and Alfred Spector, "Uniting File Systems", *UNIX Review*, March, 1989.

[Morris85] Morris, et. al, "Andrew: A Distributed Personal Computing Environment", *Communications of the ACM*, March, 1986.

[Morris88] Morris, James H., "'Make or Take' Decisions in Andrew", *Proceedings of the USENIX Technical Conference*, Dallas, February, 1988.

[Neuendorffer91] Neuendorffer, Thomas, "ADEW: A Multimedia Interface Builder for Andrew", Proceedings Multi-Media Communications, Applications and Technology Workshop, Sydney, Austrailia, 1-2 July, 1991.

[Neuendorffer90], Neuendorffer, Thomas, "ATK + 8859 = Multi-Lingual Text and Mail - A Study in Expanding the Andrew Toolkit", Proceedings of the EUUG Autumn 1990

Conference, Nice,France, October 1990.

[Palay88] Palay, et al., "The Andrew Toolkit: an Overview", *Proceedings of the USENIX Technical Conference*, Dallas, February, 1988.

[Rose86] Rose, Marshall, and John Romine, "The Rand MH Message Handling System: Users Manual", University of California, Irvine, 1986.

[Rosenberg87] Rosenberg, et al., "An Overview of the Andrew Message System", *Proceedings of SIGCOMM ’87 Workshop*, August, 1987.

[Rosenberg90] Rosenberg, et al., *Multi-media Document Translation: ODA and the EXPRES Project*, Springer Verlag, 1990.

[Scheifler87] Scheifler, Robert, and Jim Gettys, "The X Window System", *ACM Transactions on Graphics*, Vol. 5, No. 2, April, 1987.

[Schicker89] Schicker, Pietro, "Message Handling Systems, X.400", *Message Handling Systems and Distributed Applications*, E. Stefferud, O-j. Jacobsen, and P. Schicker, eds., North-Holland, 1989.

[Sherman90] Sherman, et al., "Building Hypertext on a Multimedia Toolkit: An Overview of Andrew Toolkit Hypermedia Facilities," *Hypertext: Concepts, Systems and Applications -- Proceedings of the European Conference on Hypertext*, (INRIA, France), Cambridge University Press, edited by A. Rizk, N. Streitz & J. André, November 1990, p. 13-24.

[Sherman91] Sherman, et al., "Allocation of User-Interface Resources in the Andrew Toolkit,", *Proceedings of the International Conference on Multimedia Information Systems*, (Singapore) McGraw-Hill, January, 1991.

[Swick88] Swick, Ralph, and Marck Ackerman, "The X Toolkit: More Bricks for Building User-Interfaces or Widgets For Hire", *Proceedings of the USENIX Technical Conference*, Dallas, February, 1988.

[Taylor87] Taylor, Dave, "The Elm Mail System Users Guide", Hewlett-Packard Laboratories, 1987.