Patricia Seybold's Office Computing Group

Editor-in-Chief
Judith S. Hurwitz

INSIDE

EDITORIAL
Page 2

OSF: The Product Company: *To compete against other software companies, OSF will have to learn how to market its products.*

ANALYSIS
Page 14

Unix Systems Laboratories *joins ACE* • ***Sequent:*** *the systems integrator* • ***Tivoli's*** *Wizdom: object oriented systems management, a key component of OSF's DME.*

UNIX IN THE OFFICE

Guide to Open Systems

Vol. 6, No. 10 • ISSN: 1058-4161 • October 1991

OSF's ANDF

The Key to Shrinkwrapped Software?

By Judith S. Hurwitz

IN BRIEF: OSF's ANDF could prove to be a significant technology for helping achieve shrinkwrapped open software in the future. We predict that this technology will take about three years to become widely understood and implemented by system and compiler vendors. We believe that the technology is well designed and will isolate hardware and operating system dependencies from application software.

Report begins on page 3.

OSF: The Product Company

Can It Learn to Compete?

When I first heard that the Open Software Foundation (OSF) was going to select a network management infrastructure, I suspected that it would be an overwhelming undertaking. After all, user requirements for network management in a distributed computing environment are complex. The outcome of the DME proposal was both an excellent choice of technologies and an illustration of sophisticated thinking by the selection team.

When I spoke with the selection team members, I was impressed. Even before evaluating any technology, they spoke to members of past evaluation teams and incorporated the suggestions into their own evaluation process.

LEARNING FROM EXPERIENCE. The OSF team learned a lesson from the DCE team's experience: It did not approach the DME request expecting to discover a single technology that would meet all the requirements. Thus, the team took on the challenge of looking at component technologies not necessarily as they were presented, but as they could be utilized. Therefore, pieces of technology developed for one purpose suddenly fit neatly into another slot.

THE SYSTEMS INTEGRATION CHALLENGE. But the work of DME is only beginning. Having identified the component technology that can do the job, OSF must now begin the long and difficult task of systems integration. It is no wonder that, at the announcement of the DME selection, OSF spent most of its time announcing availability of DCE source code. The fact that DCE is now available is a notable achievement. What is even more encouraging is that we will begin to see products based on the DCE technology come to market from companies such as Transarc and Groupe Bull.

BUT CAN OSF MARKET? With Motif and DCE completed and DME in the starting gate, OSF's challenges will begin to shift. OSF has proven that it can discern user requirements and build complex solutions from available technology. Now, it must prove its worth as a product and marketing company, a much harder challenge. Technologies like DCE and DME will have to compete on the open market with products from companies like Sun Microsystems and Microsoft, which understand pricing and marketing. OSF's dilemma is that it was never set up to market its own technology. That job was supposed to be assumed by its member companies. However, even if member companies are dedicated to using OSF technology, they are much more likely to push their value-added on top of products like Motif or DCE than to push the underpinnings. After all, why would a company want to promote the fact that its technology is the same as that of its competitors?

The reality is that, in the long run, OSF will not be able to sit back and wait for IBM or Digital to push as hard for OSF technology as Microsoft pushes its own products. OSF will have to learn—perhaps the hard way—to be a product-driven competitor. It is ironic that, in the end, OSF's rival is not Unix International after all.

UNIX IN THE OFFICE

Editor-in-Chief
Judith S. Hurwitz

MCI:
JHurwitz
Internet:
jhurwitz@mcimail.com

Publisher
PATRICIA B. SEYBOLD

Analysts and Editors
JUDITH R. DAVIS
DAVID S. MARSHAK
RONNI T. MARSHAK
JOHN R. RYMER

Art Director
LAURINDA P. O'CONNOR

Sales Director
RICHARD ALLSBROOK JR.

Circulation Manager
DEBORAH A. HAY

Customer Service Manager
DONALD K. BAILLARGEON

Patricia Seybold's
Office Computing Group
148 State Street, 7th Floor,
Boston, Massachusetts 02109

Telephone: (617) 742-5200 or
(800) 826-2424
Fax: (617) 742-1028
MCI: PSOCG
Internet: psocg@mcimail.com
TELEX: 6503122583

Unix in the Office (ISSN 0890-4685) is published monthly for $495 (US), $507 (Canada), and $519 (Foreign) per year by Patricia Seybold's Office Computing Group, 148 State Street, 7th Floor, Boston, MA 02109. Second-class postage permit at Boston, MA and additional mailing offices.

POSTMASTER: Send address changes to *Unix in the Office*, 148 State Street, 7th Floor, Boston, MA 02109.

OSF's ANDF

The Key to Shrinkwrapped Software?

Shrinkwrapped Software in a Heterogeneous World

If the world were perfect, there would be only one computer language that met the needs of all application developers, one operating system, and one scalable architecture. It would be possible for software developers to ship source code without fear of losing control of their software. The potential monetary power of mass market was realized for independent software vendors (ISVs) with the advent of the original Intel PC and DOS. But the dream of having shrinkwrapped applications in the Unix world is far from a reality. Ironically, what made Unix popular among developers was the fact that, because it was not tied to a binary, developers could manipulate the operating system to add the functionality needed for specialized markets. As the price point for high-powered RISC workstations continues to tumble to the $3-to-4,000 mark, vendors are finding that Unix-based RISC workstations will compete directly with PCs. Therefore, vendors selling these systems are eagerly looking for the magic formula that will turn on the applications spigot. This report will look at a technology from the Open Software Foundation called Architecture Neutral Distribution Format (ANDF), which is intended to help provide shrinkwrapped software for a multiplatform and multi-operating system environment.

Prediction: ANDF Will Be Successful in Three Years

Based on our analysis of the technology, we predict that within three years ANDF will emerge as a legitimate methodology for creating portable applications. Why three years? We believe that systems vendors will have to take the lead and provide support, education, and training to ISVs so that this new technology implementation becomes well-understood and well-trusted as underlying technology.

Can One Architecture Dominate?

Over the past five years, the workstation and server vendors have begun vying for a new-generation mass market for their technology. The stakes are incredibly high if one or more platform dominates. In the past year, the industry has begun to consolidate around six primary platforms: MIPS, SPARC, Motorola, Intel, Hewlett-Packard's HP/PA, and IBM's RS/6000. Proponents of each of these platforms are looking for the magic key to unlock the mass market potential. For example, Sun Microsystems is attempting to populate the world with SPARC-based systems by licensing the hardware specification and widely licensing its operating system. Sun's recently announced Solaris, a packaging of Unix with networking and tools, is a follow-on to this strategy. The ACE Initiative, which focuses on the MIPS RISC processor, has the same goal. Likewise, IBM and Apple's attempt to gain momentum with IBM RS/6000 and Apple's software is an attempt to win the coveted role of the leading merchant chip-based platform. All of these approaches are, at least in part, based on the notion of an Applications Binary Interface (ABI), so that the operating system is tied to the hardware binaries. Although there will be higher-level programming interfaces, they will not take the place of the operating system tied to the hardware. If we expected that one of these architectures would dominate, the problem of developing applications that could hit a mass market would be simplified. But it is becoming increasingly evident that a variety of hardware architectures will survive and thrive. Therefore, since each platform will have its own binary interface, it remains difficult for developers to port their software from one platform to another. Even if there were one binary interface, each version of Unix is sufficiently different to require that developers spend an incredible amount of time porting code. One ISV we spoke with mentioned that 70 percent of the company's development time is spent in porting—not developing.

Shrinkwrapped Software in a Heterogeneous World

Users are driven toward the concept of open systems because they want to have freedom of choice. It is ironic that this freedom is hampered by software portability problems. Users who want to purchase their platform of choice (either because of price or support) are frustrated that a particular application might not be readily available on the selected platform. A single organization might have as many as five or six different types of hardware and versions of the operating system. If that user wishes to implement the same application across all these boxes, then he or she will need a different version of the application for each platform. It makes the job of administration of a heterogeneous environment more difficult. In a distributed heterogeneous environment, users will increasingly wish to use a license server to distribute an application across a network to all users. If a different version of the application is needed for each platform, distribution of software becomes cumbersome. If users continue to be frustrated because the applications are not ported to their platforms, they will turn away from an open systems approach to purchasing technology.

The Single Compiler Dream

One way that the industry has tried to solve the shrinkwrap software dilemma is to find a single language compiler that the industry could agree upon for all platforms and all applications. But there is no simple answer to this problem. Over the past 20 years, developers have invented new compilers aimed at solving industry-specific problems (i.e., Cobol for business and Fortran for scientific programming). Hardware architectures have been invented with the specific aim of finding new ways of solving performance issues. Operating systems have been developed or enhanced to provide elegant solutions that leapfrog the competition. Every few years, some computer scientist comes up with a new plan to make these differences less important to applications developers. Many attempts have been made, for example, to produce compilers that will meet the needs of every application in every area. The development of PL/1 was intended to be a general purpose language for both business and scientific applications. It soon became apparent that it was too complex for most programmers to deal with. Ada was initiated by the U.S. government as the mandated language to meet the needs of all embedded applications development. Probably the most successful language in terms of portability has been the C language. If developers adhere to the ANSI C specification, then applications portability is much easier. Even with the use of C, ISVs still have to account for differences in the operating system and the underlying hardware platform.

The ANDF Premise: Isolating Software from Hardware

The latest organization to try to tackle this problem is the Open Software Foundation (OSF). OSF was founded as a revolt against the attempt by AT&T and Sun Microsystems to control the future of the Unix operating system and to provide a binary interface to that system. Therefore, it was not surprising that OSF should look for an answer to the binary interface between the hardware and the operating system. In April 1989 (less than a year after its formation), OSF announced that it would propose an alternative to a binary format which it called Architectural Neutral Distribution Format. In many ways this was the most controversial of all of OSF's requests for technology (RFTs). This problem has been researched for more than 20 years, and no workable solution has come to light. In addition, there is considerable skepticism within the industry regarding a neutral format as a practical solution. In essence, OSF began a fishing expedition to see if this long-wished-for goal of isolating applications from the underlying support structure could be realized. Needless to say, the ANDF requirements are very difficult to achieve. (See Illustration 1.) Is ANDF a concept whose time has come, or is it a Tower of Babel? Is it another attempt to find the one solution to all problems that is bound to end in failure? In this report, we will present the concepts behind the winner of the ANDF selection process. We will explain how it works and predict its viability.

When OSF sent out a call for technology, it asked for the following requirements:

Hardware independence. ANDF requires that the technology defer machine-dependent features (i.e., storage allocation, alignment, and size).

Protection of the developer's source code and proprietary information.

Minimal performance degradation of an application.

Extensibility to other programming languages (ANSI C is part of the original requirement).

Consistent application behavior on all platforms.

Support for hardware-specific features (i.e., user interface, networking features, operating system interfaces).

The ability to distribute data files along with application programs.

Debugger and library support.

Availability of test suites.

Portability of producer source code from one hardware architecture to another.

Backward compatibility of future installers with previous versions of the ANDF specification.

Illustration 1. The following is a list of the ANDF Requirements

Key Role of Posix and XPG

Before we explain exactly how ANDF works, it is important to explain the relationship among, the IEEE Posix standard, and the X/Open Portability Guide Level 3 (XPG3). To exist in the ANDF world, an application must first be written to an operating system that conforms with the Posix and XPG3 application programming interfaces. OSF recognizes that some areas are not yet specified in either Posix or XPG. For these cases, OSF will provide a portability guide for ISVs. If ISVs write to these APIs and to the forthcoming ANDF portability guide, then their applications should be portable to a variety of hardware and conforming operating system environments. The first implementation of ANDF will require conformance with 1003.1 Posix and ANSI C X3.159-1989. ANDF does provide a mechanism to encapsulate dependencies on functionality not in these specifications. Therefore, even if ANDF never achieves its final goal of a shrinkwrap standard, it may impress upon the industry how much can be gained by conforming to standard APIs.

Components of ANDF

ANDF technology is based on a language compiler. Every compiler consists of two components: a producer and an installer. The producer is code that creates ANDF from the application; the installer is the mechanism that binds the code to hardware and operating system, thus creating object code. In a traditional compiler, which translates high-level source language into binary objects, these two components are merged into one system. In ANDF, the producer and installer are separated. Therefore, the idea behind ANDF is to decouple hardware and operating systems dependencies from the application code. The benefits of such a concept are straightforward. An application developer could write an application without worrying about which platform it would run on. A differentiator of the

selected ANDF compiler is a special facility called a *token*. A token is similar to a macro in a programming language. Its purpose is to help isolate hardware dependencies.

The Organization behind ANDF

Originally, 24 organizations submitted technology for the ANDF RFT in April 1989. Of these, 15 submissions made the initial cut. By May 1990, OSF's evaluation team had whittled the choices down to 4, which were all based on the concept of a compiler intermediate language (IL). Unlike a conventional compiler, which produces object code that links application code with the operating system and underlying hardware, a compiler intermediate language produces an intermediate representation of the application that is not operating system or hardware specific. But, as with a conventional compiler, the source code is sufficiently changed so that it is no longer readable by humans. Three of the submissions were based on modifying existing intermediate representations to fit the RFT requirements. These submissions were from Hewlett-Packard in partnership with the University of Virginia; Peritus International (now Lucid, Incorporated), and Siemens Nixdorf Information Systems. A fourth submission, by the Royal Signals and Radar Establishment (now called the Defense Research Agency, or DRA), proposed a different approach. Unlike the other submitters, the DRA, a U.K. government research institute, had been researching applications portability for more than five years. Therefore, its submission was based on original research that was substantially different than the rest.

The TDF Specification

The specification proposed by DRA was based on the first five years of a long-term research project that coincidentally aimed at solving the problem outlined in the OSF RFT. TDF, the name of this intermediate language, wasn't designed with a particular language syntax in mind. Rather, it researched the task a language was designed to accomplish and tried to generalize that job. Therefore, the TDF design was intended to support whatever any language needs to do. At the same time, it was designed so that these concepts could be applied to existing languages. If new languages or concepts are added, then the specification can be extended to accommodate additional features.

A Structured Approach

DRA tried to develop a compiler intermediate language with the design goal of making it independent of the underlying hardware or compiler. DRA's long-term goal with the TDF technology is to allow it to be used with ANSI C, C++, Fortran 77, Cobol, Pascal, Ada, Modula2, Common Lisp, and Standard ML. The initial implementation of TDF that DRA submitted to OSF includes C. Other languages, such as Cobol and Fortran, could be supported but haven't yet been implemented. Therefore, the specification met many of the requirements in the proposal. Unlike the other intermediate language representations, TDF, which is also a binary bit stream, allows for more structured control over bits and bytes. It is based on a tree-structured and hierarchical data organization. This design makes it possible to have more granular control over the way data is stored and manipulated. In a conventional intermediate representation, information is stored as a binary stream of bits and bytes. In a tree structure, it is possible to have modules that can be moved around and manipulated based on each branch in the tree structure have an identifier.

TDF's data structure provides an abstract syntax for programs. In other words, it includes enough information to allow efficient machine code to be generated from it for a wide variety of computer architectures. A TDF data structure representing program is encoded into a linear stream of bits. This encoding is decoded at installation time. The arrangement of information makes it easier to understand how to optimize the code.

The Role of the Producer

Within ANDF, the producer is the principal tool used by the application developer. For ANSI C, OSF will develop the specification for the producer which it will license to either a system vendor or a compiler vendor (that might add value). OSF will provide a reference implementation that vendors can work with. A producer is very much like a compiler front end. It performs syntax- and semantic-checking of the language and generates an

intermediate representation of the source. The ANDF code is then distributed to end users. Through the installation process, the producer code is linked with the hardware and operating system.

The producer is invoked the same way as a compiler, giving command line options and names of source files to process. The producer analyzes the source to verify that it conforms to the syntax and semantic constraints of the language. ANDF modules are generated that represent the original application. The producer never makes target-dependent assumptions. However, the design is flexible enough so that a producer can generate ANDF from source that is explicitly target specific.

The Differences Between C and ANDF

C Code:

```
#include <stddef.h>

main () (

 size_t i;

i= 0;
```

The following ANDF is generated:

```
var_no_init ((false, true),

        tag(4),

            INTEGER (apply_token (token (0), (token(1))),

        assign (impossible,

            obtain_tag (tag (4)),

            change_var (apply_token (token (0), (token (1))),

                make_int(token (3),0))))
```

Where:

token(0) is "convert"

token (1) is "size_t"

token (3) is "signed_int"

Illustration 2 The differences between how code appears when written in the C language and when written with ANDF. Notice that, in the ANDF version, a named token is used instead of initializing the integer to the number 0. The rationale for the ANDF version is that it simplifies high-level optimizations.

Shrinkwrapped Software in a Heterogeneous World

The Role of the Installer

The installer is produced by the system vendor for a given CPU architecture. Therefore, responsibility for the installer resides with the organization that controlls the operating systems and associated applications environment. Taking a particular application from the producer onto a user's hardware requires an installer. The installer will complete the compilation process and create executable binaries that the user can run directly on the platform. The installer performs the same function as the back end of a compiler, including storage allocation, code generation and optimization, and that of a linker, binding ANDF code to native libraries.

By separating these two compiler components (the producer and the installer), the application, when processed by the producer, is isolated from the platform-specific components that are implemented within each installer. The only exception occurs if the operating system has been enhanced and the installer has likewise been enhanced to support these features.

The process of uniting the producer code with the installer happens when the user initiates the installation process. Another component of ANDF, called an *installation manager*, controls this process. The installation manager will map the token to the libraries of token definitions and can resolve any duplicate token names. It has the capability to prompt the user to set up the application based on his specific environment. The user may be asked, for example, to identify printers, graphical interfaces, even languages (i.e., English, Kanji). By the end of the installation process, an executable binary file is created.

The Power of the Token

As discussed above, TDF is a compiler intermediate language—an intermediary between the binary code and the source code. Other intermediate languages have been developed. Some of the better known include p-code (developed at the University of California) and U code (developed at Berkeley). Ada provides a standard intermediate representation called Diana. The reasons for implementing an intermediate language vary greatly. In the case of ANDF, the compiler intermediate language was used to allow the producer to be free of hardware or operating system dependencies. In the case of TDF, the token encapsulates the hardware dependencies, thus isolating the application code from the target-dependent code.

The use of tokens is what makes TDF different from other intermediate compiler languages. A token can defer a hardware-specific call until installation. Second, a token understands the rules of the language, and can, therefore, enforce semantic rules. It is much more powerful than a macro. In a programming language a user might use a macro to represent a long string of code that might be stored in a library or other file. A macro is usually resolved when the application is compiled. A token, on the other hand, is resolved in the installation process. Each token has a unique name or tag that is a variable. Therefore, a programmer could wait before assigning a value that might depend on the underlying architecture. A tag identifies the token. One resulting benefit for ISVs is that they can ship a smaller piece of code to customers. A tag naming a token can be assigned by the system or by an application developer. A developer might use a token in a number of interesting ways.

An Example of the Use of a Token

Let's take a simple example. A developer may have English and French versions of help messages for an application. Under a typical scenario, the developer would compile separate versions of that application for French and for English speakers. With the token, a different approach is possible. The developer could create a token (we'll call this Token 152) that represents the help messages. So, in the application, no help messages appear, only a reference to something called Token 152. In a separate library, the developer writes two modules: the French help messages and the English help messages. This library is shipped with the application and is used by the installer. When a user begins to install the application, he will be prompted for the language he wants his messages to appear in. The installation manager will translate Token 152 into the right version of the help message.

Here's another example using C. In C, you don't know if the Char type is signed or unsigned. Therefore, the installer has to handle this. With ANDF, the types can be substituted at the target during installation. In this way, the token won't know its characteristic until install time.

In essence, a token will allow an application developer to delay making a decision that will have operating system or hardware dependencies. For example, one hardware platform may have a special way of handling floating point operations. The application writer can use a token in place of platform-specific code. At installation time, the token is resolved based on the operating system and hardware variations. In reality, any function that a system can perform can be represented as a named or tagged token. An application developer as well as a system provider can use tokens to create special cases. In some respects, one can think of tokens as a way of defining objects. At the producer end, the application can use a container object called a token. The system does not care what is contained in that object. When information is applied to that object or token at installation time, the object inherits the behavior of the system it is now linked to.

If an ISV decided to use ANDF to build an application, he would do the following: The application builder (the developer) constructs the ANDF distribution package from ANDF object modules, native code modules, datafiles, and installation procedures. The ANDF application package is distributed to users this way. The user then purchases a version of the application in ANDF and runs the ANDF installer to put the application onto a platform. The installer completes the processing that is necessary to turn the original source code into an executable binary. The installer calls the native linker to combine binary objects and native libraries. Linking may also be deferred to run-time, as is commonly done with ABIs and shared libraries.

A Complex Migration

ANDF technology is a complex undertaking. There are many issues that will come into play before the success or failure of ANDF is known. It includes some unfamiliar concepts, such as tokens, and it will take developers and system vendors time to understand how to use them and take advantage of their power and potential. We suspect that even if ANDF itself doesn't become a well-accepted answer to the shrinkwrapped software dilemma, the concept of encapsulating specific pieces of code into tokens may become a viable future compiler concept.

Even after the industry begins to understand what ANDF could potentially do for ISVs, it will take time for a developer to trust this new way of distributing software. Why should an ISV trust that a new technology will magically make much of the drudgery of porting an application from platform to platform disappear? Will the ISV be willing to support customers who have obtained the application through an ANDF compilation? How do you trace the source of an error? Clearly, OSF will have to prove that ANDF works.

OSF's Core Deliverables

In order to gain the trust of the ISV community, OSF will have to provide a variety of tools including debuggers and verification test suites. OSF promises to offer the following producer tools:

- The ANDF Technical Specification.
- An ANSI C Producer, which converts C language source code into ANDF language.
- An ANDF-to-ANDF Linker, which links ANDF modules prior to distribution and removes external references where possible.

A Method of Development and Distribution

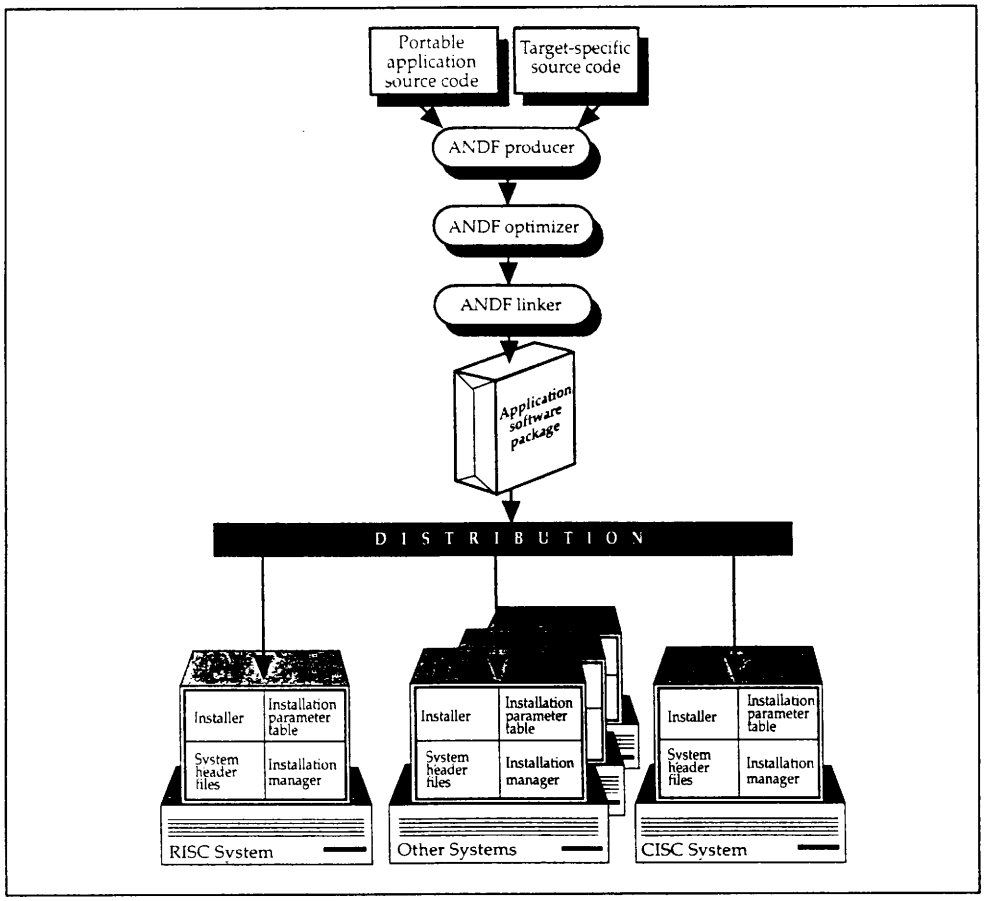


Illustration 3. This illustrates the way ANDF works. An application is separated into its target independent components and its target dependent parts. It is then run through the ANDF producer. ANDF intermediate code is created, then optimized, if desired, and finally linked. At the back end, there is an installer for each hardware platform that combines the application code with the platform-specific code. At installation, token definitions are resolved.

- An ANDF-to-ANDF Optimizer.
- An Installation Manager.
- An ANDF Librarian, which creates an ANDF library prior to distribution.
- Three reference installers for SCO Open Desktop for the Intel 386, MIPS with the Ultrix operating system, and VAX with the Ultrix operating system. DRA, which developed the technology, will contribute two ports: one for SPARC running SunOS, and another for the Motorola 68000 running HP/UX. Each of these installers will include support for Debuggers.

In addition, OSF intends to encourage third-party vendors to provide additional tools, such as portability checkers and other application development tools. It will look to third parties for additional language support, including C++, Cobol, and Fortran.

Underpinnings for ANDF

Before ANDF becomes accepted, it will have to be tied to underlying technology. Most notably, it will have to be tied into DCE and DME. DCE will provide the networking

underpinnings, while DME will provide the mechanism needed to distribute ANDF code. For example, the license server component will be an important mechanism for distributing applications across a heterogeneous network. Indeed, a technology such as ANDF is not intended to work in isolation.

ANDF Means New Programming Techniques

In order for ANDF to succeed, programmers and applications developers will have to make dramatic changes in the way code is developed. Some of these changes in technique are already beginning to happen. Developers have learned the hard way that, to survive in a world filled with multiple operating systems, users interfaces, and hardware platforms, they have to write modular code. They are beginning to learn the benefits of writing to the Posix specification. The joys of portable C code are becoming well known in programming circles. But ANDF means even more changes for developers. They will have to take more dramatic steps in separating out any operating system or hardware dependencies from their code. Learning to use the powerful token technology will be a challenge.

Performance Issues

One of the key requirements of the ANDF RFT is that the addition of ANDF technology not noticeably impact performance. The target range for performance degradation in the RFT was no more than 5 percent. Current estimates with the TDF technology are as low as 3 percent. DRA believes that, through the effective use of tokens, this technology could prove to perform better than conventional compilers. The one area where ANDF adds time is in the installation process. Because a considerable amount of translation of target dependencies happens during installation, it may take longer than conventional installation techniques.

Alternatives to ANDF

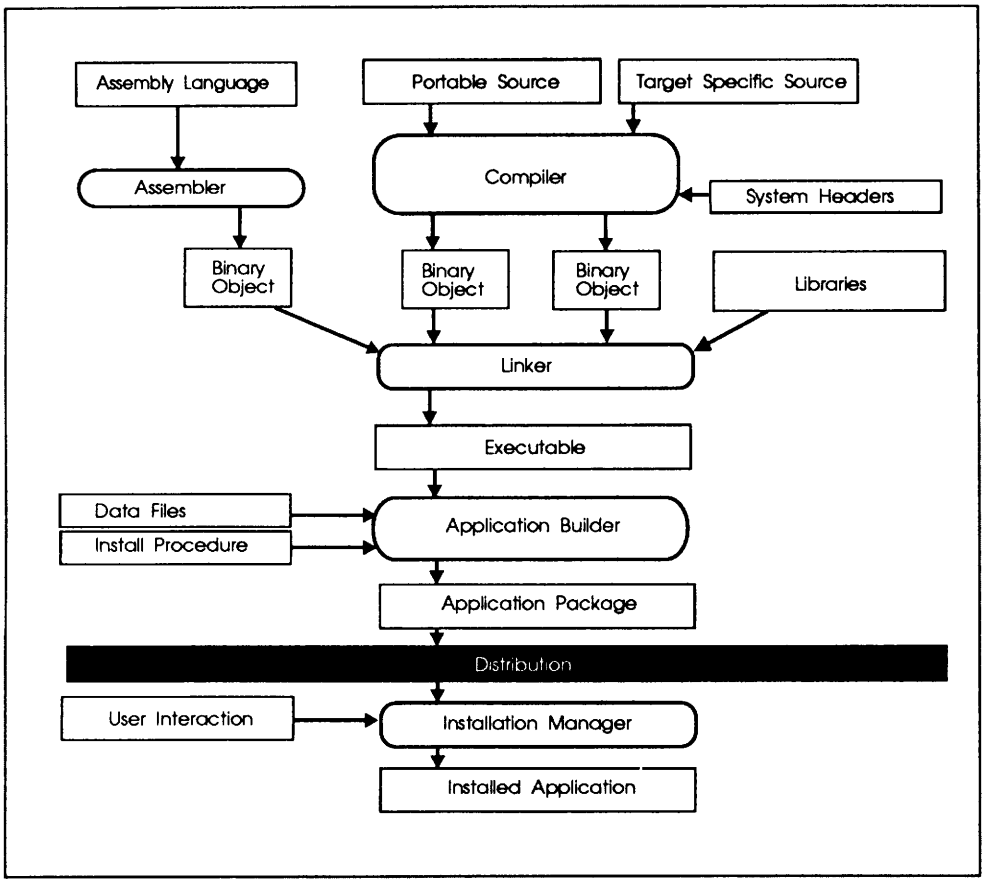
There are other ways that can be used to achieve applications portability. Each method has its pluses and minuses. Some of the techniques include:

- Source Code. Source Code is human readable and is the easiest way to move an application from platform to platform. The main problem with source code is that it affords no protection to the software developer.

- Application Binary Interface (ABI). An ABI defines the interface to the operating environment, instruction set, and binary software conventions. The benefit is that an application can run without recompilation on any system in the hardware family that is running the same operating system. While an ABI can be an effective strategy within one architecture, it is problematic for developers who wish to move their applications to another platform. The concept of the ABI may be strengthened by combining an ABI with ANDF. In this way, developers would write to an ABI within one architecture and move between architectures using ANDF. The developer writes the ANDF application, while the underlying installer can support the specific ABI. ANDF enforces the ABI's programming interface.

- CD-ROM. Being able to distribute applications via CD-ROM makes sense. Some in the industry have suggested that a developer would put multiple implementations of applications on one CD-ROM to help solve the application distribution challenge. However, this does not change the need for developers to recode components of their applications for different hardware and operating system platforms.

Comparison between Traditional and ANDF Development Environments

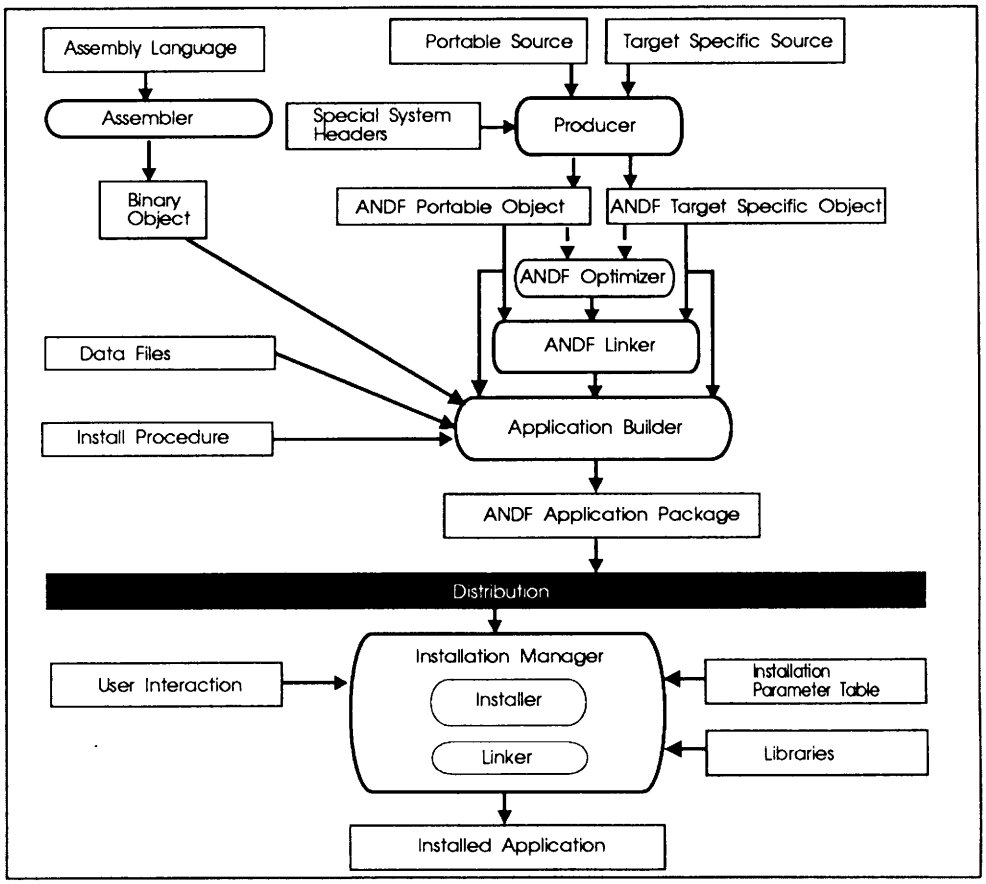


The way applications are developed with traditional techniques is very different from the way they are developed with ANDF concepts and technology.

Conclusion

ANDF is compelling technology. It offers the potential for application developers to more easily port their applications to a wide variety of platforms with minimal effort. Machine dependencies are moved from within the application to token definitions. The most important aspect of ANDF is the fact that it proves that portability is not impossible. It should pique interest among system and application providers in studying and testing this technology. In reality, the most difficult aspect of ANDF may not be a technical hurdle but one of confidence. System vendors will have to implement installers for their platforms, they will have to work with key software developers to help them experiment with this technology, and they will have to test and verify the quality of an application ported through ANDF. Therefore, they will have to assume the initial responsibility of providing support to customers. If ISVs are to trust such a new concept, they need verification that they will not lose business if something unexpected results from an ANDF-based port. But, like all bold moves, ANDF deserves a chance to prove that this concept could revolutionize the way software is written and ported.

Comparison between Traditional and ANDF Development Environments (continued)



We expect that ANDF will not enjoy a quick adoption curve. Rather, it will be a slow, cautious experiment. There may be some unanticipated stumbling blocks. All scenarios have not yet been tested. Until there is more data and more experience, the industry will reserve judgment. Therefore, it is important that system vendors take the lead. At the same time, users who have suffered because the software they need is often not available must actively encourage experimentation with ANDF. If ANDF does turn out to be the solution to applications portability, then the small risk will generate significant results. ◐

Next month's *Unix in the Office* will address:
The SQL Standard.

For reprint information on articles appearing in this issue,
please contact Donald Baillargeon at (617) 742-5200, extension 117.

Open Systems: Analysis, Issues, & Opinions

UNIX SYSTEMS LABORATORIES

Unix Systems Labs Flexes Its Muscle

Just as AT&T and NCR are beginning to get used to their new marriage, their offspring, Unix Systems Laboratories (USL), is working hard to forge its own path through life. Now that it is no longer protected under the vast AT&T umbrella, USL is trying hard to look and act like a competitor. One of the first signs of a new USL was its recent announcement to join the ACE Initiative. Why should USL join with an organization that has pledged to implement OSF/1 and Microsoft's NT operating system? Simple. It may be better to get a nice slice of the pie than to be left with an empty plate. USL was taking a lot of heat from MIPS OEMs that were determined to use System V.4.

A Good Move

Perception is always at least 85 percent of reality (I could round it off to 90 percent, but 85 sounds more scientific). Therefore, even though at this juncture ISVs will be choosing among three different operating systems to write to, there is a growing perception that the Unix wars may be ending. Perhaps, in the future, these organizations under the auspices of ACE will actually find a way to have a common software layer that isolates the ISV from the differences in these operating systems. But I believe that dream will be realized in the future. Ironically, it may be that ANDF, the technology discussed in this month's feature, might be at least part of the problem created by ACE.

The Ramifications of USL and ACE

When the ACE initiative was formed, I was skeptical. I wondered whether anything concrete could come out of a group that seemed to want to provide a piece of the action for anyone not called IBM or Apple. There were two operating systems, two user interfaces, and several I/O interfaces. MIPS promised that users would be shielded from the differences. But initial reports suggest that users may have to recompile each piece of code repeatedly to move it from one version to another. We understand that the Digital/SCO work on the combined Ultrix and OSF/1 operating system is progressing slowly. One bright spot in the initiative was the inclusion of Silicon Graphics 3-D IRIS GL graphics engine.

With USL entering the group, the picture may begin to change. If Unix System V.4 can be modularized as USL is promising (the so-called System V Light) so that a typical user can have a fully configured system that needs only 4MB, Unix may be well positioned. Another encouraging sign is USL's agreement with Novell, which promises to become a primary distributor of System V.4 as part of the ACE initiative. Because Novell is a part owner in USL now, it has a lot to gain by seeing some success in this frenzied marketplace.

Conclusion

ACE seems to be gaining some momentum at a very interesting time in the industry. With Microsoft and IBM embroiled in a bitter war of words and deeds, it is not surprising that smaller competitors find themselves strangely on the same side. Ironically, the IBM/Apple plan to bring out a new joint technology was in part a reaction to Microsoft's participation in ACE. Now, USL's endorsement of ACE with the strong hint that it will be all right to work at some level with OSF's operating system technology may be the most important consequence of all this warfare. *—J. Hurwitz*

SEQUENT COMPUTER SYSTEMS

Sequent: Approaching Systems Integration

It is quite easy to lump all hot-box vendors together. It is also easy to think of them as simply another generation of fast-box vendors concerned only with turning inventory and little with software. But one vendor I recently met with made me realize that the hot-box industry is changing. These second-tier vendors are now in direct competition with companies like IBM, Digital, and Hewlett-Packard. Therefore, to survive,

they must move upscale in their attention to the details of systems integration and software. An example of a box vendor that seems to understand this new set of requirements is Sequent Computer Systems. Despite some notable achievements, Sequent has gotten most attention lately for its poor financial showing. The sudden downturn was caused by the failure of one of its key OEM customers, Unisys, failing to purchase as many boxes as it had originally promised. Management had planned around those inflated numbers. This was an unfortunate occurrence, especially given Sequent's interesting technology and approach. Sequent hopes that it can distinguish itself in this complex and competitive hardware market by focusing its attention on three areas: implementation of parallelization, object technology, and systems integration.

Approach to Parallelization

Sequent's claim to fame has been its ability to create a sophisticated Unix kernel in conjunction with AT&T's System V. that allowed it to develop tightly coupled, symmetrical multiprocessing systems. Sequent has used its understanding of multiprocessing and load-balancing to elevate itself into a position where it could be a premier database engine. Quietly, Sequent has been working with its key database partners, such as Oracle, to improve performance by parallelizing certain tasks so that the entire database would run faster under the Sequent engine. Typically, Sequent turns these tricks of the trade over to the database provider. Thus, Sequent would not be at the mercy of each new iteration of the database. However, Sequent will have as much as a two-year head start on the competition.

Approach to Object Technology

In a move that seems atypical for a box-maker, Sequent formed the Object Technology Partnership with some object-oriented technology vendors such as ParcPlace Systems, Reusable Solutions, Tigre Object Systems, and Versant Object Technology. Their goal is to ensure that this new breed of applications development tools will be ported in large numbers to Sequent's SMP platforms. We usually don't expect second-tier players like Sequent to take such a lead in object technology.

Systems Integration

But what impressed me the most was Sequent's work in the systems integration arena. Clearly, management has done its homework. Initially, when Sequent got into the server business, it focused on interoperability with the corporate mainframe. Its boxes found their way into departments running key applications that were not typically found on mainframes but needed to communicate with them. But, over time, Sequent has come to realize that the real opportunity for integration was not between a server and the host but between PC LANs and hosts. It discovered that 95 percent of PC LAN-to-host communication was via terminal emulation. Not exactly state-of-the-art client/server processing. Therefore, Sequent's approach has been to begin doing systems integration at a layer above the hardware. Its most recent target has been the Novell LAN environment. Sequent recently signed an agreement with Novell where it will implement a parallelized version of Portable NetWare. Running as many tasks in parallel on Portable NetWare, which Sequent made possible by multithreading the Novell kernel, greatly enhances performance. This should benefit both Sequent and Novell. Poor performance has long plagued Portable NetWare. Again, as it has done with its parallelization work with database vendors, Sequent will not own the code. Instead, it will turn it over to Novell, but with a head start in marketing this souped-up version on its own servers. Sequent doesn't stop there. It has also implemented a parallelized version of IBM peer-to-peer protocol, LU6.2, so that communication between the LAN and host can be more efficient. The advantage of parallelized communications combined with a more efficient LAN operating system means that users can rapidly access host-based information without resorting to slow terminal emulation.

Conclusion

Sequent seems to have an understanding of the raw power needed to compete as well as the sophistication to begin implementing parallelized hardware and, more importantly, parallelized software. Not an insignificant achievement. However, for Sequent to make its way in an increasingly competitive world, it will have to convince users and ISVs that it is more than another box maker out for the fastest MIPS in the world. It will have to begin to act like a systems integrator. Sequent will also have to overcome its image as a financially troubled player. Sequent seems to have the talent in place and its focus on the right technology trends for success.

—J. Hurwitz

NETWORK/SYSTEMS MANAGEMENT WATCH

Tivoli Wizdom: Object-Oriented Systems Management

Tivoli's Wizard Distributed Object Management (Wizdom) system is at the heart of the OSF DME. That fact may not result in a lick of sales revenue for Tivoli, but it is important independent corroboration of the company's technology. Wizdom is a sophisticated, object-oriented approach to systems management applications that appears destined to play a major role in distributed management.

Wizdom is a systems management service platform and applications set. It is based on a client-server architecture, with distributed servers. The first release, scheduled for availability in December 1991, will run on Sun-TCP/IP networks. Tivoli chose Unix as its initial platform because it believes Unix users have the greatest distributed systems management problems. From our experience, this is a reasonable assumption—many Novell users are just thinking about implementing the kinds of distributed environments that are typical of Sun installations. However, Tivoli plans to add support for other platforms in subsequent releases of Wizdom.

WHAT IS SYSTEMS MANAGEMENT? Tivoli defines systems management as encompassing three types of resources. They are:

- Physical resources
- Logical resources
- Collections

Physical Resources. Physical resources are the major elements of the environment, including printers, servers, workstations, gateways, and users. The primary management functions relevant to physical resources are allocation and control.

Logical Resources. Logical resources are the services and applications built atop the physical resources, including file services, print services, mail systems, and applications. the primary management function relevant to logical resources is the enforcement of policies on their usage.

Collections. Collections define the major organizational structures in a distributed environment, ranging from user groups to groups of servers and other devices to "management domains," or network segments. The primary management functions relevant to collections are monitoring and reporting.

ADMINISTRATION/MANAGEMENT USERS. One of the major problems Tivoli is seeking to solve with Wizdom is system administrator burnout. System administrators are responsible for too many administrative and management tasks to be effective as networks grow larger. And today's low-level tools make most of these tasks labor intensive. Tivoli designed Wizdom to allow administration and management tasks to be divided among three users within the typical corporate structure:

- System administrators
- End users
- MIS departments

System Administrators. For system administrators, Wizdom provides a way to automate routine tasks (such as changing user account configurations), to delegate tasks to others, and to spend more time on the definition and enforcement of policies.

End Users. When system administrators delegate functions, they will often delegate to end users. Wizdom's user interface and applications paradigms are designed to make administrative tasks, such as changing passwords and resetting printers, accessible to end users.

MIS. For MIS, Wizdom is crafted to support the creation of custom management applications, to conform with major management standards such as CMIP and SNMP, and to be secure. Authentication and authorization are built into the Wizdom framework.

WHAT IS WIZDOM? Wizdom implements a distributed object management framework to shield applications and applications developers from the details of system internals. For this reason, Wizdom is a management platform in its own right. Tivoli layers its infrastructure on top of the systems management facilities provided by vendors such as Sun and Digital—seeking to provide a common applications interface to them. Even among Unix implementations, there are many differences in base management facilities. many different systems management platforms.

Tivoli believes that a common underlying service structure will enable users to at last get away from the discontinuity problems created by today's discrete and unconnected management applications. The company's primary interest is in applications, not platforms. (See Illustration A.)

Wizdom's Architecture

Wizdom comprises three components:

- An object-oriented messaging framework
- Core applications
- Direct manipulation graphical interface

Wizdom is based on an object-oriented approach to management applications. In Wizdom, everything is an object, including the display characteristics of the object. These are objects in the sense that object-oriented programming products define objects—bundles of data, attributes, and program code that can be invoked to perform tasks by sending them a request message.

Wizdom allows new objects to be created using existing objects in two ways. First, new objects can be defined from collections of objects. Second, objects can inherit methods from other objects.

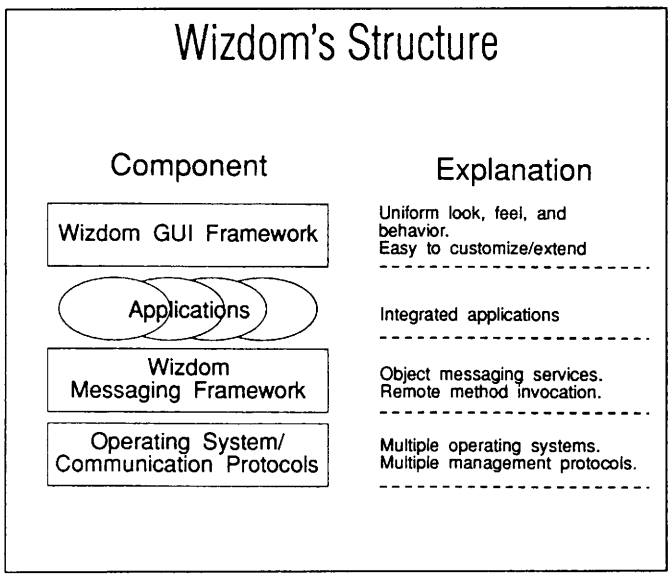


Illustration A. Tivoli's Wizdom implements an object-oriented messaging framework on top of existing systems and network management services and protocols.

Wizdom applications operate on objects representing all of the major resources, policies, and other elements in a distributed environment. An application may, for example, send a message to an object asking that object to perform a particular operation. An application might direct a user object to change its network address to a new address.

In the same way, applications can send messages to objects asking them to return information about themselves. A security management application, for example, might ask an object representing a group of users to return the names of all of its users classified as "super users." (See Illustration B.)

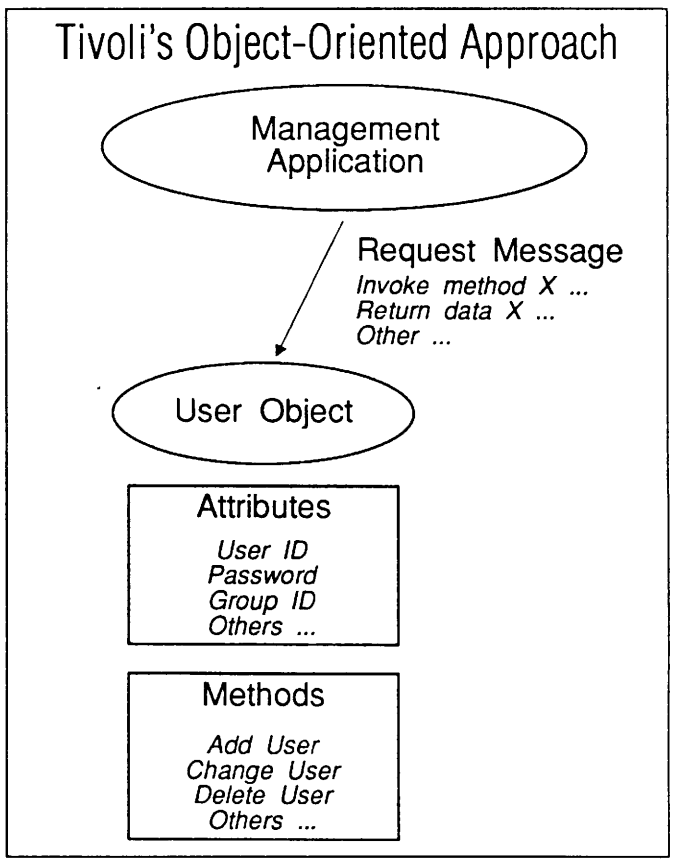


Illustration B. Tivoli's Wizdom allows elements in a distributed environment to be defined as objects capable of performing their own operations, or methods. An application directs objects to perform methods by sending them a request message. In the above example, an application could send a particular user object a method directing the object to change its network address.

Object Management Framework. Wizdom's Object Framework is a messaging, dispatching, and synchronization facility that ensures that request messages from applications are delivered to the proper object and that the response is returned to the client. The framework includes a dispatching service, a persistent store for objects, and an object manager to manage interactions between objects and the messaging framework. Wizdom allows new objects to be added to an existing environment without recompiling existing objects.

The framework appears to be similar in nature to the Object Management Group's (OMG's) Common Object Request Broker Architecture (CORBA) messaging standard. If anything, Wizdom's framework is more complete than CORBA's. Wizdom integrates security into message dispatching, for example, while CORBA does not. The Wizdom framework integrates the authorization and authentication services of OSF's Distributed Computing Environment. The two distributed object management approaches appear destined to evolve toward greater and greater commonality.

Graphical User Interface Framework. The Wizdom GUI framework is based on a set of libraries that are independent of all look-and-feel standards. In a heterogeneous distributed environment, this is the only realistic approach Tivoli can take. In its first release, Wizdom's GUI framework will provide interpreters for OpenLook, OSF Motif, and X Windows. Tivoli plans to add other interpreters in later releases.

Tivoli's GUI allows users to manipulate a distributed environment by directly manipulating icons and other graphical display elements. To create a new user account, for example, an administrator can drag and drop an icon representing a user on an icon representing a host. This action stimulates a dialogue that results in the definition of the new account. System administrators can delegate certain tasks by creating a new management domain and defining its permissions to include only the delegated tasks.

Applications Development Facilities. Despite its roots in object-oriented programming concepts, Wizdom does not force administrators to learn object-oriented programming to use the environment. Wizdom will provide a toolkit that includes classes, message templates, and other applications building blocks. Wizdom toolkits are libraries for a variety of languages.

Wizdom supports the definition of system elements as objects using a variety of programming tools. In the first release, system administration specialists can use awk scripts, shell scripts, and C programs to define objects and their methods. Tivoli can add other language options over time by simply providing libraries that provide access to the Tivoli framework.

Wizdom also includes a scripting tool for the graphical user interface.

Management Applications. Tivoli plans to ship the first release of Wizdom with several core applications. At press time, the final packaging for the first release wasn't settled. However, we expect Wizdom to ship with applications for user management (adding users, browsing the properties of user objects, moving users on existing networks, etc.), security and privilege management (setting security attributes on objects, changing user access privileges, etc.), and applications management (license verification, usage statistics, etc.).

MIGRATION SHOULD BE SMOOTH. Wizdom appears to be capable of sliding into existing environments without disrupting them. Most of the configuration required to set up a Wizdom system is completed automatically upon installation of the product. Wizdom reads existing configuration, password, and other files to create objects representing the major elements in a network.

In addition, because Wizdom can accommodate many languages—including scripting languages—to create objects, users can encapsulate their custom management scripts to work in Wizdom.

Wizdom appears also to be positioned for integration with other management applications, particularly network management products. Wizdom will operate beside network management platforms, such as SunNet Manager, and their applications. Wizdom applications should be able to interact with management applications written to a management platform API. The primary means for this interaction will be for a Wizdom object to invoke an agent in the service of SNMP, CMIP, or other protocols to obtain information or transmit instructions.

Pricing and Packaging

At press time, Tivoli had not yet determined how it will package and price the Wizdom technology. It appears likely, however, that Tivoli will sell Wizdom as a base environment with simple applications for functions such as user account management, with optional applications for functions such as security management.

The only thing Tivoli will reveal about its pricing plans is that they will be designed to make the base framework easy for users to acquire. The company's target customers are users with Unix networks of between 20 and 100 nodes.

Conclusions about Wizdom

Wizdom is impressive technology. Conceptually, we like the object-oriented approach of network management upon which it is based. The object approach is better able to cope with the inevitable complexity of a distributed environment than the protocol-based approach is. This is vital.

Tivoli also has obviously spent a lot of time creating software that will be small and unobtrusive. The company estimates that its protocols are one-tenth the size of the protocols in OSF's Distributed Computing Environment. The Wizdom framework is implemented as a series of coordinated Unix daemons.

Tivoli is a small, vendor-capital-financed company. Therefore, many users will be initially reluctant to make big commitments to it. However, Tivoli's central position in OSF's DME will probably help ensure that its APIs will be adopted by other vendors. Any vendor that adopts the DME package will be supporting Tivoli's API. Thus, users can get started building applications to Wizdom today with a reasonable assurance that the APIs will be long-lived.

Tivoli's participation in DME is not without risk. Our biggest concern is that the DME integration process will distract Tivoli from rolling out support for additional platforms. Sun is a good place to start, but there's a whole wide world of non-Sun sites out there. We hope Tivoli can move rapidly to Novell and Windows environments after establishing its beachhead in Unix. At the same time, we'd also like to see Tivoli address IBM and Digital host environments. —*J. R. Rymer*

Patricia Seybold's Computer Industry Reports

Topics covered in Patricia Seybold's Computer Industry Reports in 1991:

Back Issues are available, call (617) 742-5200 for more information.

Office Computing Report

1991—Volume 14

#	Date	Title
3	Mar.	Object-Oriented Development—A New Foundation for Software in the '90s
4	Apr.	Xerox DocuTeam—A Compelling Reason to Take a New Look at Xerox
5	May	The Battle for LAN-Based E-Mail—Lotus, Microsoft, and WordPerfect Go Head to Head
6	June	IBM OS/2 2.0—The Quest for the Desk
7	July	End-User Information Systems—An EIS for the Rest of Us
8	Aug.	The Windows Office—Evaluating Microsoft Windows as the De Facto Desktop Office Environment
9	Sept.	Unraveling the NewWave Confusion—Differentiating the NewWave Environment from the NewWave Office from Microsoft Windows

UNIX in the Office

1991—Volume 6

#	Date	Title
3	Mar.	Ingres—A Database Vendor in Transition
4	Apr.	Open CASE—Toward an Open Systems Infrastructure
5	May	Clarity's Rapport—The Designing of an Integrating Application
6	June	Uniface—Developing Database-Independent Applications
7	July	Can Digital Become an Open Software Company?
8	Aug.	Interbase Software—Extending the Relational Model to Handle Complex Data
9	Sept.	Uniplex's New Vision—A Pragmatic Approach to the Open Office

Network Monitor

1991—Volume 6

#	Date	Title
3	Mar.	Digital NAS—Services for the Distributed Computing Environment
4	Apr.	Sun Open Network Computing—Responding to the Challenge
5	May	PeerLogic PIPES Platform—Building Distributed Applications
6	June	Ellipse—LAN-Based OLTP Platform
7	July	IBM/Distributed Systems—Big Blue's Emerging Client/Server Architecture
8	Aug.	Name Services—Converging on a Two-Tier Model
9	Sept.	Common Object Request Broker—OMG's New Standard for Distributed Object Management

ORDER FORM

Please start my subscription to:		U.S.A.	Canada	Foreign
☐ *Patricia Seybold's Office Computing Report*	12 issues per year	$385	$397	$409
☐ *Patricia Seybold's Unix in the Office*	12 issues per year	$495	$507	$519
☐ *Patricia Seybold's Network Monitor*	12 issues per year	$495	$507	$519
☐ *Paradigm Shift—Patricia Seybold's Guide to the Information Revolution*	10 issues & tapes per year	$395	$407	$419
☐ *Paradigm Shift—Patricia Seybold's Guide to the Information Revolution*	10 issues per year	$295	$307	$319

Please send me a sample of: ☐ *Network Monitor* ☐ *Office Computing Report* ☐ *Unix in the Office* ☐ *Paradigm Shift—Patricia Seybold's Guide to the Information Revolution*

Please send me information on: ☐ Consulting ☐ Special Reports ☐ Conferences

☐ **My check for $_____ is enclosed.** ☐ **Please bill me.** ☐ **Please charge my subscription to: Mastercard/Visa/American Express** (circle one)

Name: _____ Title: _____

Company Name: _____ Dept.: _____

Address: _____

City, State, Zip code: _____

Country: _____ Bus. Tel. No.: _____

Card #: _____

Exp. Date: _____

Signature: _____

Checks from Canada and elsewhere outside the United States should be made payable in U.S. dollars. You may transfer funds directly to our bank: Shawmut Bank of Boston, State Street Branch, Boston, MA 02109, into the account of Patricia Seybold's Office Computing Group, account number 20-093-118-6. Please be sure to identify the name of the subscriber and nature of the order if funds are transferred bank-to-bank.

Send to: Patricia Seybold's Office Computing Group: 148 State Street, Boston MA 02109; FAX: 1-617-742-1028; MCI Mail: PSOCG

To order by phone: call (617) 742-5200

IU-1091

STATEMENT OF OWNERSHIP, MANAGEMENT AND CIRCULATION. **1A. Title of Publication:** Patricia Seybold's Unix in the Office. **1B. Publication no.:** 08949921. **2. Date of filing:** October 1, 1991. **3. Frequency of issue:** Monthly. **3A. No. of issues published annually:** 12. **3B. Annual subscription price:** $495.00. **4. Complete mailing address of known office of publication:** 148 State St., 7th Floor, Boston, MA 02109. **5. Complete mailing address of the headquarters of general business offices of the publisher:** 148 State St., 7th Floor, Boston, MA 02109. **6. Publisher:** Patricia B. Seybold, 148 State St., 7th Floor, Boston, MA 02109. **Editor:** Judith S. Hurwitz, 148 State St., 7th Floor, Boston MA 02109. **Managing Editor:** Ronni T. Marshak, 148 State St., 7th Floor, Boston MA 02109. **7. Owner:** Patricia B. Seybold, 148 State St., 7th Floor, Boston MA, 02109, **8. Known bondholder, mortgagees, and other security holders owning or holding 1 percent or more of total bonds, mortgages, or securities:** None. 9. **For completion by nonprofit organizations authorized to mail at special rates:** Not applicable. 10. **Extent and Nature of Circulation:** A. Total no. copies printed Average 12 Mos. 1600 Single Issue 1200 B. Paid Circulation 1. Sales through dealers and carriers, street vendors and counter sales Average 12 Mos. None Single Issue None 2. Mail Subscriptions Average 12 Mos. 1200 Single Issue 950 C. Total Paid and or Requested Circulation Average 12 Mos. 1200 Single Issue 950 D. Free distribution by mail, carrier or other means samples, complimentary, and other free copies Average 12 Mos. 50 Single Issue 50 E. Total Distribution Average 12 Mos. 1250 Single Issue 1000 F. Copies not Distributed 1. Office use, left over, unaccounted, spoiled after printing Average 12 Mos. 350 Single Issue 200 2. Returns from news agents Average 12 Mos. None Single Issue None G. Total Average 12 Mos. 1600 Single Issue 1200 I certify that the statements made by me above are correct and complete. (signed) Patricia B. Seybold Publisher