Patricia Seybold's Office Computing Group

Editor-in-Chief
Michael A. Goulde

INSIDE

UNIX IN THE OFFICE

Guide to Open Systems

Vol. 7, No. 3 • ISSN: 1058-4161 • March 1992

Europe's Harness Project

Integrated Technology for an Open, Object-Oriented, Distributed Applications Platform

By Joshua Greenbaum

IN BRIEF: Perhaps the greatest role open systems can play in the 1990s is promoting the growth of distributed systems, and, without a doubt, the greatest role object-oriented technology can play is in the simplification of complex programming tasks. Putting the two together is the goal of a European project, Harness. This is an object-oriented software development environment for complex distributed applications that shields the developer from hardware and operating system fundamentals and the conflicting communications and internetworking protocols they entail.

It's an ambitious goal, nothing less than the Holy Grail of open systems, and Harness' stellar cast of European hardware and software partners are determined to have a working prototype by the end of the year. The technological foundations of Harness are sound and the project is well-defined and well-underway. But Harness has a lot of ground to cover in a short time, and there is no dearth of competing standards and forthcoming technology that may offer commercially viable alternatives for much of the environment the project expects to define. *Report begins on page 3.*

Unix and Open Systems

Are They Two Directions or Just Two Perspectives?

FOR 20 YEARS, Unix held a unique position as the one portable, source-licensable operating system that was commercially viable. Its widespread availability supported advanced academic research in operating system and networking theory. As interest spread to commercial settings, Unix allowed new computer companies with new architectures to spring up almost overnight, creating new markets and filling important gaps in the marketplace. Unix served as the foundation for the emergence of an entire culture and, ultimately, for the open systems movement.

Today, the unique position held by Unix is being challenged from two directions. Microsoft's Windows/NT is challenging its property of portability, and emerging POSIX standards are challenging its position as the only basis for open systems.

Unix Systems Laboratories doesn't have sole ownership of Unix. We don't mean that it doesn't own the Unix technology, which, of course, it does, but that, to a large extent, Unix is owned by the programmers and engineers who have built careers on Unix. They have developed applications, extensions, utilities, file systems, networking capabilities, and even entire computer companies around it. For them, the emerging threats to Unix run much deeper than a simple technical or marketing challenge. They are threats to the professional identity and even the self-esteem of these individuals.

This situation is not unlike one that existed in the field of psychiatry 25 years ago when some renegade psychologists suggested that B. F. Skinner's theories of operant conditioning were superior to Freud's psychoanalytic theories when it came to treating psychiatric symptoms. These so-called behaviorists were virtually ostracized from the field when they suggested that human behavior could be modified by rearranging the environmental contingencies of reinforcement that determined the relative probability of adaptive and maladaptive behaviors. Emotions, they reasoned, weren't the cause of behavior, but were intellectual and physiological reactions to environmental contingencies. "Heresy," said the psychoanalysts, who perceived their years of training, personal psychoanalysis, and patient care as devalued by these new theories.

The analogous situation today is that the Unix traditionalists say that open systems are based on the Unix operating system and its interfaces, while the revisionists say that open systems are based on standard interface definitions that should not be dependent on where Unix is headed and that can be implemented by a variety of underlying technologies. Just as the belief system of the psychoanalysts was challenged by the behaviorists, so too is the belief system of the Unix community being challenged by this new breed. What should we call them? Are they the open systems community? The POSIX community? The standard interface community?

This is an important question that needs to be resolved if users are to have a clear path to fulfilling their requirements. Unix and non-Unix technologies have to be placed in proper perspective. Unix and its derivatives carry assumptions about the way systems and networks should work that may or may not be "right."

In the past, the focus of this newsletter has been Unix, its evolution, the applications that run on it, the vendors who market it, and the organizations that influence it. We are now questioning whether *Unix in the Office* needs to look at broader issues, and we are looking to its readers to tell us. Your completing and returning the enclosed survey will help us decide what the focus of this newsletter should be and what it should contain. Please take the time to fill out the survey and fax or mail it back to us.

Incidentally, as a result of empirical research that showed behavioral therapy to be more effective than psychoanalysis in certain types of disorders, behaviorism has moved into the mainstream of mental health. Classical Freudian psychoanalysis has not disappeared. It plays a different role today, and many analysts have accommodated Skinner's ideas in their practice.

UNIX IN THE OFFICE

Editor-in-Chief
Michael A. Goulde

MCI:
MGoulde
Internet:
mgoulde@mcimail.com

Publisher
PATRICIA B. SEYBOLD

Analysts and Editors
JUDITH R. DAVIS
ROSEMARY B. FOY
DAVID S. MARSHAK
RONNI T. MARSHAK
JOHN R. RYMER
ANDREW D. WOLFE, JR.

News Editor
DAVID S. MARSHAK

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.

Europe's Harness Project

Integrated Technology for an Open, Object-Oriented, Distributed Applications Platform

Harness Overview

The Harness project lies at the center of a series of concentric circles of European economic, technological, and political ambition. On the outermost rim is the European Community (EC), whose ambitious charter, as yet unfilled, is to unite the vastly different countries, economies, and cultures of Europe into a single political and economic entity. Within the EC lies the Commission of the European Communities, the administrative and policy-making body that is chartered with fulfilling the EC's mandate. Inside the next circle is Directorate-General XIII, which is responsible for Telecommunications, Information Industries, and Innovation—the technological heart of much that the EC stands for. Finally, within DG-XIII is the European Strategic Programme for Research and Development in Information Technology, better known by the acronym ESPRIT. One of the largest government-run pure and applied research programs on the planet, ESPRIT consumes approximately $1,000 per minute and consists of over 1,000 R&D teams, organized into a loose consortium of corporate sponsors and academic researchers whose task is to perform applied and pure research in microelectronics, information processing, business systems, computer-integrated manufacturing, and other technology-driven subjects.

Integration Is Harness Focal Point

While Harness is one of over 100 research projects within ESPRIT's Advanced Business and Home Systems group, it is very much at the core of ESPRIT's and the EC's hopes for European leadership in distributed systems. Integration is the watchword at Harness: The consortium itself is involved in little fundamental research, drawing the lion's share of its technological and applied software base from numerous on-going ESPRIT projects and outside standards, such as the Open Software Foundation's (OSF's) Distributed Computing Environment (DCE) and IEEE POSIX. Its work has a broad scope, its goals are ambitious—working prototypes are due in by the end of the year, commercial products are expected in 1994—and the potential impact is enormous. If it fulfills its goals, Harness stands to influence a European open systems market that, in many ways, is ahead of that in the United States. As a result, Harness could also influence the U.S. market, which is itself proceeding towards distributed applications technology, albeit without a central, unifying, Harness-like body to guide it.

Prototype Is Harness Research Goal

Influence does not translate into commercial product development in the language of ESPRIT, however. It is important to note that Harness is restricted to research by ESPRIT's guidelines, though such research can extend into the so-called industrial prototype phase, equivalent to an alpha test stage. While its goals clearly state that Harness will concern itself with developing such an industrial prototype, the actual productization must be done by the private sector.

Overall Scope and Objectives

Harness started in December 1990 and will end in May 1993, with a total effort projected at 74 person-years and total funding at 10.5 million European Currency Units (ECUs), or approximately $12 million. Half of Harness's funding comes directly from the coffers of ESPRIT, and the other half comes from the research organizations themselves.

Overall Scope and Objectives

Harness Players and Project Goals

Harness has 16 member companies drawn from the cream of Europe's Information Technology (IT) industry. Led by Cap Gemini Innovation, the R&D arm of French software and services giant Cap Gemini Sogeti, Harness includes researchers from:

- British Telecommunications plc
- Bull SA
- Siemens Nixdorf, Inc.
- Bell Northern Research Europe
- Grupo APD SA, a Spanish software and integration house
- Volmac Nederland BV, a Dutch software and services company
- Architecture Projects Management Ltd, a U.K.-based company that is currently commercializing a major software component of Harness
- KAPSCH, an Austrian data communications developer whose specialization in the IBM world plays a key role in the project

Besides its commercial partners, Harness also draws on considerable academic and research expertise from organizations including:

- INESC, a research and development lab in Lisbon
- Datamont S.p.A, the research arm of the Italian industrial concern Ferruzzi Group
- ENST, the French National College for Systems and Telecommunications
- INRIA, a French national research and development lab
- Communications Networks Research Group, a Greek research house
- Trinity College, Dublin, a hotbed of European object-oriented research
- The University of Antwerp

The basic design of Harness calls for providing an object-oriented applications development environment for supporting wide area and local area networking of heterogeneous platforms, including proprietary as well as open systems. Its designers expect Harness to be used in applications including office automation, corporate information systems, government administration, engineering, and scientific research. However, the platform is neither intended for the exacting needs of the real-time market (except in a limited fashion) nor is it intended to satisfy the security requirements of the military or defense establishment.

Targets Multiple Audiences

The targeted user for Harness is the applications programmer. Facilities and services are provided for the system administrator and systems programmer, particularly for the distribution of resources, tuning and error recovery. Harness's support for large, distributed systems running on wide area networks makes its targeted user environment major corporations and government agencies.

Distributed Object Model

While the fundamental design calls for an object-oriented, open environment (with Unix-like operating systems considered as an important, though not exclusive, criterion), legacy technologies figure prominently in Harness's specifications. The project's goals are not limited to providing a platform for developing new object-oriented applications. There is a conscious effort to allow for a wide range of existing applications, written in procedural code, to be encapsulated within the Harness object model and distributed within the Harness system.

Integration of Technologies and Standards

Technology integration is at the heart of Harness's work. In addition to drawing on the work of several existing or recently completed ESPRIT projects, DCE, and POSIX, the project is also tracking standards from ISO, IEEE, CCITT, the Object Management Group (OMG), and X/Open. Harness plans to make itself open to a range of existing technical specifications including remote procedure calls, object and procedural languages, communications protocols, and interface specifications. Higher-level services, such as database, user interface, and the applications themselves, will be kept independent of the underlying Harness base. Likewise, the base operating environment is largely immaterial to Harness; as long as the environment is POSIX compliant, Harness plans to be able to function within its domain.

Remote procedure call (RPC) specifications included in Harness are Hewlett-Packard's NCS, Netwise's RPC, and the OSI RPC. Distributed services to be supported include Sun's ONC and the ANSA specification from the ISA/ANSA ESPRIT project. Object-messaging and transfer technology, similar to the OMG's Object Request Broker (ORB), will come from ISA/ANSA as well. Harness's developers are considering, though not necessarily promising, to support the OMG's ORB.

Moving from Design to Implementation

As it approaches the final year of the project, the Harness team is beginning to move from the drawing board into the prototype phase. Until now, very little Harness code has actually been written, though there is a considerable body of working code available from the existing ESPRIT projects and other open systems technology sources. The Harness team will wind up its work by May 1993.

The timing of implementation is more than just the caprice of a bureaucratic mind. Technology like DCE is still not widely distributed, or even well-known, within the applications community. Many of the POSIX standards are still in a state of flux, and object-oriented services and data models are still on drawing boards both in Europe and the United States. The project leadership is convinced that, by the time Harness winds up its work, its base technologies will be better known in the market and the skill required for development will begin to emerge. This, as will be explained later, may prove to be a two-edged sword for Harness.

Technological Underpinnings of Harness

Several key technologies are being integrated into the final working Harness software platform. Several are basic foundations for the project, including COMANDOS, the COMANDOS VMI, and ISA/ASNA.

Object Management and Language Foundations

COMANDOS. Construction and Management of Distributed Open Systems, or COMANDOS, is an on-going ESPRIT project that began in 1986 with the intention of showing the feasibility of using an object-oriented approach to distributed applications development.

The basic components of COMANDOS include:

- An object-typing model, which allows the use of and interaction among most truly object-oriented languages
- A virtual machine, which provides the basic mechanisms for masking the differences among underlying systems architectures from the object-oriented language

The major programming languages of Harness include C++ and Eiffel, an object-oriented language developed by Interactive Software Engineering of Goleta, California, as well as the COMANDOS Object-Oriented Language. The COMANDOS project specifies tools for

both object program development and distribution management. Its target operating environments are the OSF Mach kernel, the Chorus microkernel, and generic Unix.

VIRTUAL MACHINE INTERFACE. The virtual machine and its interface, known as the COMANDOS VMI, are key elements of the ESPRIT project that will find their way into the Harness environment. The virtual machine handles persistent object storage, controls distributed computations, and manages transactions and network communications. The VMI contains language-specific run-time environments that handle calls from the program to lower levels of the operating environment and vice versa.

While COMANDOS is a good framework for the Harness project, it lacks two important features: support for object environments that are not based on object-oriented languages (i.e., encapsulated procedure programs) and support for the wide area networks targeted by Harness. Adding these two features will be Harness's major enhancements to the COMANDOS work.

COMANDOS will continue its development work independently of Harness, and its research and development phase will terminate at the end of 1992.

ANSAware Supports Distributed Functionality

The Integrated Systems Architecture for the ODP (Open Distributed Processing) project, also known as ISA/ANSA, was started as a feasibility project intended to provide a technological understanding of distributed systems. While the project is still on-going, the results of the initial phase are available under the product name ANSAware, which is sold in source code form by Harness partner Architecture Projects Management, Ltd.

ANSAware, which is a set of software and tools for the development of distributed systems, is a major piece of Harness's distributed functionality, the other being the OSF Distributed Computing Environment specification. ANSAware is also based on object-oriented techniques and provides an interface definition language that allows applications to interoperate across a distributed environment.

The main additions to ANSAware that Harness has undertaken include support for pure object-oriented languages and persistent object storage as supported by the COMANDOS environment.

OSF DCE Services Also Supported

OSF's DCE provides many of the same services as ANSAware, though the two differ primarily as to how their object trader or broker mechanisms are presented. Harness's support of DCE is partly fueled by the acknowledgement that DCE has garnered widespread support as a workable, although complex, technology. It's important to note that most of the major developers of Harness are also OSF members; thus, there is an imperative to make good on each company's investment in OSF.

Harness will incorporate the distributed functionality provided by DCE, though the project plans to make the environment more of an applications programmer tool than a systems programmer tool.

SOS, Delta-4, and DOMAINS Components

Three other ESPRIT projects figure in Harness, though to a lesser degree than the projects mentioned above.

SOS. SOS was a demonstration project undertaken by French Harness member INRIA to prove the feasibility of developing large, distributed systems using object-oriented technology. In addition to proving the point that a language like C++ is appropriate for a project

SOS, Delta-4, and DOMAINS Components

like Harness, SOS has contributed technology that supports object migration and has provided Harness with a toolkit of objects that handle communications and other protocols.

Delta-4. Another distributed systems project, Delta-4, will make major contributions to Harness in the areas of clock or timing services, security, communications, and fault tolerance.

DOMAINS. DOMAINS is an on-going ESPRIT project that will bring Harness important expertise in the field of managing open distributed systems.

The Harness Architecture: Four Conceptual Layers

How will all this existing technology integrate into a single Harness platform? The basic Harness model can be described architecturally as consisting of four conceptual layers built on top of a POSIX-compatible operating system (see Illustration 1).

Harness Structural Architecture

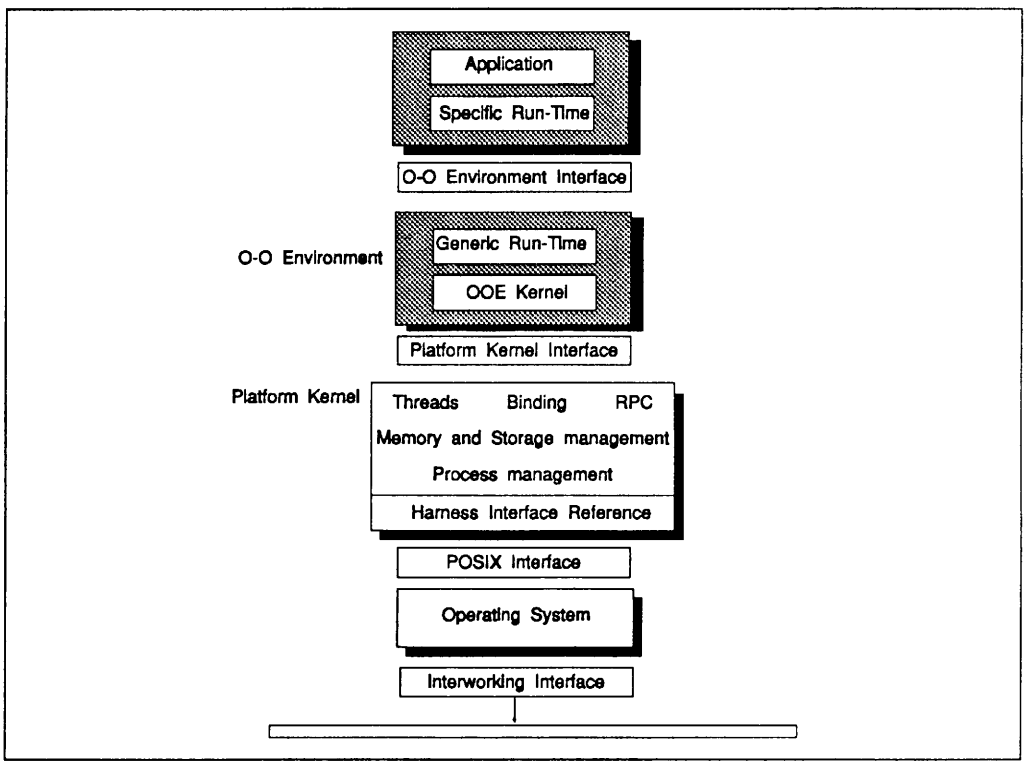

Illustration 1.

Top Layer: Object-Oriented Environment

The top layer, the object-oriented environment (OOE) provides an interface that supports object language programs running on the Harness platform. This layer is the major contribution of the COMANDOS project, and, as such, its interface specification is virtually identical to the COMANDOS VMI, with additions that encapsulate procedural programs and support wide area networks.

The OOE provides a number of facilities to support object environments, including object and transaction management as well as the control of distributed computations and networking.

The Harness Architecture: Four Conceptual Layers

Second Layer: Platform Kernel Interface

Below the OOE sits the platform kernel interface (PKI), which acts as a generic interface to the distributed environment, whether that is DCE or ANSAware. Calls to the PKI are made either from object programs via the OOE or directly, in the case of encapsulated procedural programs. Procedural programs may also call the PKI in a more elegant manner either by embedding distributed program calls or by using an interface definition language that is still being defined. The intended result is to allow a Cobol or other procedural program to interface with the PKI in an object-oriented manner.

Third Layer: Harness Interface Reference

Driving the PKI is a specification called the Harness Interface Reference (HIR). This is the mechanism by which the PKI handles—invisibly to the programmer—interactions between remote applications. Harness is able to provide access to remote procedure calls and threads by using HIRs. The HIR will allow Harness to specify both DCE and ANSAware initially, and will allow for the support of other distributed systems, including, potentially, the distributed specifications of SAA.

Bottom Layer: Operating System Interface

The base, or the lowest level within the Harness specification, is the operating system interface, called OS in the acronym-heavy jargon of Harness. This last layer is based on POSIX and will make POSIX calls to the operating environment resident on the hardware platform or platforms in which Harness applications run.

Potential for a Fifth Layer

The Harness specification also recognizes the potential need for a fourth interface at the bottom of the pile--an interworking interface, or IW, that would define standard communications down to the cable level. The main reason for this specification is to guarantee against the incompatibility of otherwise similar communications protocols. For systems that differ in their underlying communications technology, the interworking interface would provide a mechanism for arbitrating these differences and controlling the interaction among transport protocols. This is qualified as being an option, and will not necessarily be required of all systems. Harness recognizes that most applications will be able to address all their communications needs through the upper three levels.

Harness in Operation

Using this conceptual model as a guideline, the basic operation of the Harness model holds few mysteries for those familiar with the concept of a layered, distributed computing system.

Calling the Environment

In the ideal Harness environment, applications, whether object oriented or procedural, enter the Harness environment by one of two ways: either exclusively through the OOE, in the case of pure object programs, or by calling elements of the OOE, PKI, and/or OS directly. This second method will be used by less-disciplined object programs and procedure language programs, the latter by making procedure calls to each of the interfaces, using an interface definition language. Harness's designers would prefer to see programmers who are developing object-based programs maintain an orthodox development methodology and restrict object programs to dealing exclusively with the OOE. But Harness recognizes that a less-than-ideal programming world exists, and that many application programmers may want to control specific systems services in a more direct manner. Therefore, Harness allows all programs to make calls directly to the OOE, PKI, or OS levels. Regardless of how a call is made, the interface at each level acts as a virtual environment that masks the distribution of lower-level services required by the application.

Distribution of Tasks Across the Network

The distribution of tasks across the network is the responsibility of the PKI. PKI both manages the distribution of applications and provides its own virtual environment, which, in turn, makes systems calls compliant with specific RPC syntaxes. Those calls then execute through the operating system interface across the network as one would expect of a dis-

Harness in Operation

tributed system, functioning across the network using whatever RPC syntax is available. In this way, the PKI offers a transparent interface to the writer of distributed applications.

The HIR Brokers Object Requests

The HIR functions as the invisible traffic cop of the Harness model, ensuring that applications can call the resources they need anywhere within the distributed environment while hiding the underlying mechanism from the programmer. Because of the need to mediate the services of disparate distribution technologies such as those provided by DCE and ANSAware, HIR was developed as a separate and unique specification that can traverse the range of platforms supported by Harness. For the programmer, the HIR is an abstraction. He or she merely calls a pointer that, in turn, navigates the network in search of the service provided.

Harness depends on the HIR in particular because the promise of a truly distributed system requires the ability to move services around a network dynamically, taking advantage of available resources in what amounts to ad hoc resource allocation. As such, HIR acts as a relative address mechanism: The programmer calls a service, automatically tagged by Harness with an HIR which allows the PKI to find the service and generate the correct remote procedure call.

The HIR specification is designed to deal with a considerable amount of exception-handling in order to track movable servers and to access alternate resources if a given server is unavailable. Therefore, in addition to a relative address, the HIR is also able to specify where to go if resources are unavailable, and the specification allows for multiple HIRs for redundancy and fault tolerance. Likewise, the HIR specification can provide a single client with access to multiple servers in the event that a given service is distributed.

Applications and Services Supported

APPLICATION MODULES. The rich set of specifications in Harness is intended to support three basic types of applications, each of which can traverse one or more layers of the platform as it performs its tasks. The first comprises the applications modules (AM), which, as their name indicates, constitute the object-oriented and procedural applications, mostly end-user oriented, that live at the top of the Harness three-level model. For example, an application module might find and access spreadsheet cells or check mail messages for selected senders.

GENERIC SERVICES. Harness also defines what it calls Generic Services (GS), which are adjunct applications that act primarily as extensions of or supplements to AM programs like the spreadsheet cell access service described above. The GS category includes services such as database access systems, system management programs, X.500 and X.400 services, X Window, and software tools. These GS applications can be either local or remote to the AM, and can themselves be distributed from platform to platform.

PLATFORM KERNEL GENERIC SERVICES. Platform Kernel Generic services (PKG), can also reside locally or remotely. The archetype PKG is the trader or object broker, which may be a generic technology like the OMG's ORB but must nonetheless be tied in closely with the specifics of one or more of the layers in the Harness model. As such, PKGs are allowed to have special access to interface routines and internals that would be masked by a GS or AM program.

The Interaction of AMs, GSs, and PKGs

The basis of the Harness environment is formed by the interaction of AMs, GSs, and PKGs. A standard interaction model that is essential to their activity defines the interfaces that they must have in order to work within Harness. The designers of Harness have spent considerable effort refining this specification, which requires that both object and procedural pro-

able effort refining this specification, which requires that both object and procedural programs adhere to it in order to remain compatible with one another. The basic terminology used to define the interaction model allows for requester and recipient objects, which map quite well to the client/server world. Asynchronous, synchronous, and deferred synchronous requests are all supported.

Architecture in Operation

Conceptually, the location of a particular module on the network is largely irrelevant as long as the interfaces are well-defined and uniform, although, by their nature, PKGs must be resident on the platform for which they are designed.

From the perspective of the software modules themselves, communications and interoperations simply entail making calls to the appropriate module or modules in a manner conforming to the interface specification, which serves to hide Harness's multi-layer architecture from the application.

A practical way of looking at this interaction is to consider Illustration 2. AM 2 is the main application, while AM 1, GS 1, and GS 2 are local applications whose relative (and specific) locations are already known to AM 2. The main application also requires the services of nonlocal applications AM 3 and AM 4, known only by a relative address. PKG 1 is, in this case, the trader or broker application, local to the platform on which AM 2 resides, that mediates the network navigation for local applications.

Application Interaction Model

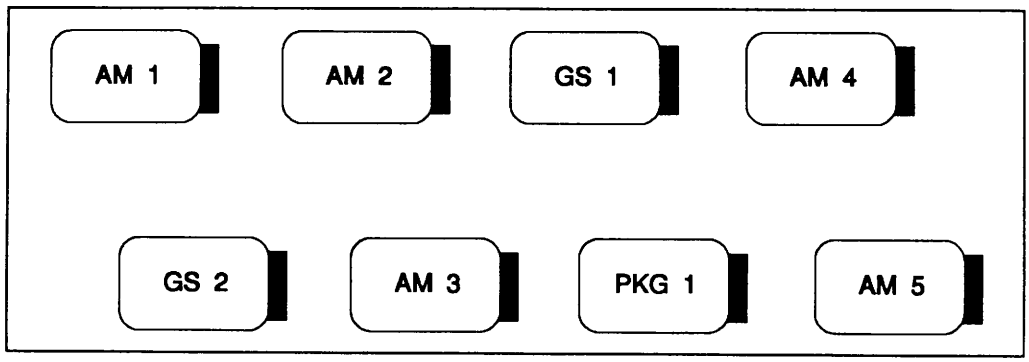

Illustration 2.

The sequence of events that constitutes the execution of AM 2 is shown in the accompanying table. AM 3 and AM 4 make themselves known to PKG 1, the trader, which generates a pointer, or HIR, for each application. A sequence of requests and replies from the main application to its local services and its local application is generated. Then AM 2 looks for a nonlocal application, turning to the trader in order to process the HIR call and interconnect with AM 3. Following a request from AM 2, AM 3 itself turns to the trader in order to find AM 4, whose reply cascades through the network back to AM 2. The execution is then complete.

Practically speaking, any of the programs in the example could be written in a procedural or an object-oriented language. The interactions, in the form of requests or replies, are hidden under the veil of interface specifications that, for the procedural language, look like procedure calls, while, for the object language, they look like objects or pseudo-objects.

Interaction Scenario Example

request	AM 3 to PKG 1	registration with "Trader" module
request	AM 4 to PKG 1	registration with "Trader" module
request	AM 5 to PKG 1	registration with "Trader" module
request	AM 2 to GS 1	request to a known GS module
reply	GS 1 to AM 2	result from GS 1
request	AM 2 to AM 1	request to a known application module
request	AM 1 to GS 1	request to a known GS module
reply	GS 1 to AM 1	result from GS 1
reply	AM 1 to AM 2	Result from AM 1
request	AM 2 to PKG 1	request name of service provider from "Trader"
reply	PKG 1 to AM 2	name of AM 3 returned
request	AM 2 to AM 3	request to the newly known application module
request	AM 3 to PKG 1	request name of service provider from "Trader"
reply	PKG 1 to AM 3	name of AM 4 returned
request	AM 3 to AM 4	request to the newly known application module
reply	AM 4 to AM 3	result from AM 4
reply	AM 3 to AM 2	result finally returned to AM 2

Putting Harness to Use

Harness anticipates three types of users, each with differing needs and therefore differing access to the system. Their interaction with Harness is portrayed in Illustration 3.

End Users Ultimately Benefit

End users are the ultimate beneficiaries of Harness, but their access to the system will be primarily through the use of application modules. Few, if any, underlying services will be available or even necessary to the majority of end users.

System Administrators Have Full Access

System administrators, who will bear the responsibility for establishing the Harness environment, allocating resources, and tuning the system, will have access to the full range of services. The system administrator will thus be able to manage the configuration and security of the system, as well as design new applications that will fall within the scope of GS and PKG programs.

Heaviest Users Will Be Programmers

Although the system administrator has greatest access to Harness, the application programmer is envisioned as the primary user of the system. However, the applications programmer's access to services will be controlled through specified interfaces, as befitting a system that attempts to enforce a level of discipline on applications in order to ensure portability and interoperability. Therefore, the programmer is required to be familiar with a supported language and a set of development toolkits, and will have to be aware of the availability of system resources and their relative addresses. An appropriate user interface technology will also be part of the programmer's repertoire. Significantly, knowledge of remote procedure call specifics will not be required, nor will the differences in operating systems, network communications, and hardware specifics be part of the programmer's toolkit.

Putting Harness to Use

General Scheme for the use of Harness

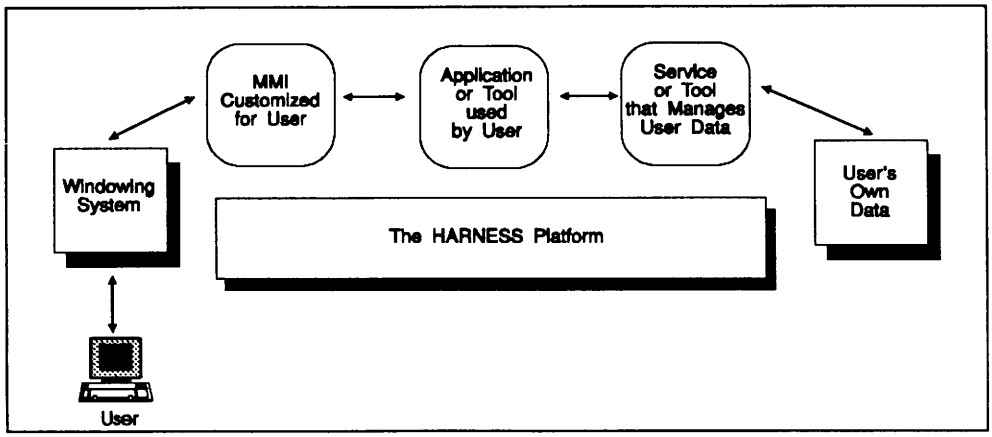

Illustration 3.

Behind-the-Scenes Services. Most other services will be hidden from applications programmers, if they so wish, though programmers are free to break the rules. The interface language that links program modules will function without direct programmer manipulation, as will the internals of the HIR.

For other services, for example database access, the programmer is required to be expert in the database system without needing to know the specifics of accessing remote servers; access is expected to be encapsulated in the Harness design.

So, while the programmer will be relieved from having to deal with the programmatic complexity of Harness, a well-designed application will have to be made with the Harness model very much in mind.

Choice of Programming Environments

Designers always describe an ideal programming environment for their systems, and good designers always allow for the fact that programmers often shy away from following even the best of rules.

Such is the case with Harness, which will specify the use of modified versions of C++. One such version, called C**, has been developed by Harness partner Trinity College as part of the COMANDOS project. The other is EC++, from Harness partner INESC. Both are cleaned up, extended versions of C++ that are designed to restrict or discipline the C++ user to following orthodox object-oriented programming practices. Harness designers are particularly concerned with the ability within C++ to violate standard encapsulation practices, as well as the ability to directly manipulate pointers and other program internals that are outside the pale of acceptable object-oriented programming techniques.

In the real world, however, C++ will be used, and Harness's designers anticipate that vendors will offer C++ compilers complete with the appropriate libraries for accessing Harness facilities and services. Since Eiffel is used in the COMANDOS project, it will probably be made available under Harness. Provisions for other object-oriented languages, such as Smalltalk and Ada, are not included in the Harness specification, though the system is designed in such a way as to allow additional languages as needed.

User Interface Work Remains

One of the unfinished aspects of Harness is in the user interface. The objective is to have an intermediary user interface system that would specify a set of criteria for user/application interaction and act as a mediator between the application front end, the (non-Harness) user

User Interface Work Remains

interface system, and the Harness platform. The rationale behind such a system, called the User Interface Management System (UIMS), or Man-Machine Interface (MMI), is to provide a uniform look-and-feel as well as compatibility with commercial windowing systems such as X Window, Windows 3.x, Presentation Manager, Motif, and OpenLook, all of which Harness expects to support.

UIMS Separates Interface and Logic

The UIMS or MMI stipulates that applications be written so that data manipulation is kept programmatically separate from the part of the program that performs presentation and interaction functions. Code for this latter functionality would be responsible for interfacing with the UIMS, providing relatively easy changes in the interaction and presentation without changing the program internals.

This UIMS specification is recognized by Harness as more of a pipe dream than a reality, and, assuming that no such UIMS system can be found or developed, Harness has specified criteria for an applications programming interface that allows for UIMS-like support for the applications programmer.

Distributed Applications Programming Requires Discipline

The wealth of technological support for distributed systems provided by Harness places the onus on the programmer to take into account a new set of disciplines when programming in this environment. This means that more will be required of programmers than the simple mastery of the Harness APIs. Understanding the theoretical workings of distributed systems will be a necessary first step before well-disciplined programs can be written.

An example is dealing with the possibility of a system failure somewhere in the network. The potential for adjunct applications and services to be distributed virtually anywhere an HIR can find them means that programmers have to account for the special cases that arise when an application requests another application or service on a system that is unavailable. While the HIR mechanism will attempt to redirect the request to a new or replacement server, the application must be able to elegantly recover from the very real possibility that no replacement will be available. This will require error-checking discipline on the part of the applications programmer, who must provide the appropriate exit routines in the case of a failure at some node in the system. This may become particularly burdensome for the programmer who is migrating legacy applications into the Harness system.

Reprogramming Existing Applications

On the technical side, programming existing applications to run in Harness requires little more than programmer discipline and a knowledge of the principles of object programming and the Harness platform. Harness has made compatibility with existing applications a major criterion, and considerable technical detail in the specifications is devoted to this.

Redesigning Legacy Applications

On the practical side, making a legacy application run efficiently in a distributed environment like Harness may require considerable reworking of program internals, particularly with respect to the architectural separation of a previously monolithic program into a distributed application. The requirement for error-checking in the case of nonexistent resources is particularly acute in these older programs.

Of course, there is no requirement to redistribute legacy applications, and indeed the programmer can merely encapsulate programmatic routines and entry points to be part of a distributed environment, even if the entire functionality remains local to a single machine.

Decisions Have to Be Made

Although Harness will provide the environment to support existing, legacy applications, the developer will have to weigh the value of a full-scale migration or rewrite into the Harness environment. How those decisions will be made is left up to the developer–Harness merely provides the tools to make it possible.

Harness Project Plans for 1992

Harness Project Plans for 1992

Current plans for Harness call for the development of two working prototypes by the end of 1992, one based on ANSAware, the other based on DCE. This appears to be a political compromise more than a technical one. Although the decision to develop both prototypes has been made, and interoperability is a cherished goal of the Harness project, there is admittedly still a lot of work to be done to see if and how the two environments will be made to work together.

Bringing ANSA and DCE Together

The main problem between the two environments exists at the all-important trader level. ANSA and DCE specify different traders and different naming conventions, making them, in the main, incompatible. One of the Harness partners is looking into supporting the DCE RPC mechanism under ANSAware, and plans to have the same OOE on both systems makes compatibility a functional possibility. But the consortium has a long way to go before a solution is available.

Obviously, deployment by the end of the year depends on more than just the integration of the technology Harness inherited from other ESPRIT projects. OSF's DCE, which had fallen behind its original development schedule, needs to be not only fully functional and available, but also well-known to the development community that would be expected to put its precepts into practice on a Harness platform. The implementation of the POSIX standard into working operating systems is also just beginning, and experience in their use and deployment is limited at this time.

Factors Affecting Harness's Outcome and Impact

Harness, while well-thought-out, well-intentioned, and well-designed, suffers from a number of limitations, some of which are technical, some market-driven, and some political. None is fatal or even life-threatening, and most are more the result of factors outside the control of Harness than of some oversight or neglect on the part of the design team. But the potential problems must be considered within the broader context of the commercial viability of Harness.

Ambitious Task—Aggressive Time Frame

While Harness is based on years of solid research and development across Europe and the United States, its ambitious goals belie the monumental task that lies before it. Technology integration on the scale Harness is now undertaking is a difficult task, particularly in an open systems world that, by necessity, must often place compatibility, not technical excellence, at the fore. The track records of the OSF and the OMG may be considered as models for this type of effort: Delays are more the norm than the exception, and compromise must be constantly weighed against expediency. While there is every reason to think that Harness will be able to accomplish its task, the gaps in technology that Harness must fill are not insignificant, and it may very well follow the usual pattern in open systems development and experience delays either in technical development or eventual productization.

Integrating and Supporting Evolving Technologies

Also militating against Harness is the fact that much of the technology it wishes to integrate is not frozen but fluid, and is still undergoing change. COMANDOS and ISA/ANSA are ongoing projects that will not close until the end of the year, and the technology now being used from these projects within Harness is a snapshot of work taken in 1991. In particular, the relation between today's Harness and tomorrow's COMANDOS, while close, is not written in stone.

Other specifications outside of ESPRIT and the open systems movement, like IBM's SAA, can neither be ignored nor planned for, and yet Harness must account for the SAA world if

it is to fulfill its promise of being both open and accessible to the huge body of proprietary and legacy (hence, IBM) systems.

Immaturity of Object Standards

Of critical importance is the fact that the entire object-oriented world is still very much in its infancy. The OMG, after some delay, has an object request broker and is now working on an object model and other baseline object technologies. And, despite the unprecedented unanimity within the U.S. computer industry regarding the ORB specification, Harness will most likely stick to its own and primarily support ANSAware. The object projects now underway jointly between Apple and IBM and between Sun and HP are also potentially significant sources of important fundamental research and industry standards that will not figure in the Harness project.

Commitment and Competition

The OSF and OMG examples are also important with regard to eventual product development. While member companies such as Cap, Bull, and Siemens Nixdorf have been heavy participants in Harness and Harness-related development, and while Harness was undertaken in part to bring the results of pure research into the commercial world, there are no firm commitments from any participants regarding product plans. Meanwhile, the market churns ahead outside the Harness world, and the possibility exists that alternatives to Harness may crop up that could threaten its adoption. At that point, one must question whether any of the Harness companies would be so Euro-centric in its strategic thinking that it would choose a distributed applications platform that had little hope of becoming a worldwide standard.

This points to a fundamental flaw in the development of open systems technology that, while not the fault of Harness, nonetheless may make its future success as a widely-adopted standard difficult to achieve. Because the program is so broad in scope and was initiated before many of the efforts that are now contributing technology, the designers have had to work with early snapshots of technology and code from many different sources as they have become available. And, while the developers of Harness are moving relatively swiftly to accomplish their task, the component technologies of Harness are independently evolving at the same time.

The Open Systems Paradox

These facts produce the classic open systems paradox, which weighs heavily on the Harness project. On the one hand, Harness cannot afford to delay its implementation to include the newest technology and standards from various corners of the open and not-so open systems world because doing so would jeopardize vendor support. On the other hand, if it moves ahead and has product that is deliverable by May 1993 as planned, new technologies or standards may appear by then that are more attractive than those which Harness has selected for inclusion.

Are Transparent Distributed Open Systems Feasible?

Even if Harness should make its debut on time and fully formed, the notion of transparently distributed open systems platforms exists largely in the abstract. While much can be done in a laboratory setting to simulate the real world, the monumental integration task now underway will only be complete when real-world applications are applied to Harness's theoretical base. The Harness team is well aware of this fact. To Harness's credit, several working systems have been spawned from the ESPRIT program that provide examples of subsets of Harness's functionality. One example is a large distributed astrophysical database used in the United States by NASA and developed using ANSAware. Harness can also draw on the experience of a distributed banking project called Bank '92 when looking for working examples of the kinds of systems it is developing. But, as Harness continues towards an expected working prototype by December, it is unlikely that it will remain immune from the vagaries endemic to charting new territory.

Security Issues Pose Challenge

Finally, there is the issue of security, a problem that troubles not just Harness but every distributed open systems effort. On a purely technical level, Harness's specification will provide the necessary hooks to support a secure system, but no such system will be included in the working prototypes. Harness will base its security technology on DCE and other offer-

ings but will be dependent on the underlying platform, which, in the case of Unix, implies that system security may be less than robust. In general, security issues are particularly problematic for distributed systems, where the physical distribution of resources makes it virtually impossible to put up a "cordon sanitaire" around the kind of WAN that Harness envisions serving. To date, the Harness developers acknowledge that the state of the practice does not provide for an acceptable security system beyond what Unix and POSIX envision. Obviously, Harness will in no way restrict security provisions for database and applications software, but the underlying Harness system will not be as secure as even its designers would like. This is an issue that is critical to the success of Harness-like systems, and its resolution rests well beyond the scope of the Harness project.

Conclusion

Harness is good technology, well-designed and worthy of taking its place as a model for distributed systems. And its operative goal of working from an existing technology base, even when forced to make hard choices that may seem politically motivated and fixed at a less-than-ideal point in technological development, is probably the best way to develop open systems today.

Can Harness Attract Support?

As with any specification that hopes to be a standard one day, the future of Harness will live and die by the rate at which it is adopted. And its adoption will be very much a measure of its timeliness in a market that has many competing notions of how such an object-oriented, open and distributed software platform should look.

Adoption as a standard, however, doesn't have to be restricted to its member partners and other, strictly European, open systems vendors. Therein lies a measure of hope for Harness as an international, not just European, standard. While ESPRIT has a measure of fortress-Europe mentality, in that its goals are European technology for European companies, the EC's definition of a European company is broad enough to include most major U.S. open systems vendors. IBM, Digital Equipment, HP, and Sun, among others, are sufficiently European to partake of the fruits of Harness, should they so desire.

Harness in the Marketplace

It remains to be seen what will happen once Harness ventures out into the marketplace. While Harness has specified an extensive set of conformance criteria for systems based on the specification—a move that is unusual in the research-oriented ESPRIT world—the one aspect of an eventual product that Harness can't and won't mention is marketing. And yet marketing is ultimately the point on which the success of the project will turn.

Next month's *Unix in the Office* will address
The X Window System Today

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

Open Systems: Analysis, Issues, & Opinions

FOCUS: MICROSOFT CORPORATION

Microsoft Takes the Offensive in Open Systems

Open Systems as Defined by Microsoft

The advertisement in the *Wall Street Journal* and *Business Week* was headlined: "If everybody says they have open systems, how can you tell who really does?" The ad went on to say, "... we believe an open operating system sets a consistent standard for everybody," and "which gives you and your company freedom of choice in finding the best way to fulfill your personal computing needs." In other words, Microsoft is committed to open systems *as it defines them.*

In the Microsoft model, an open system has published APIs and fosters the development of third-party applications and enhancements. Microsoft develops those APIs, getting input from developers in the process. While this definition of open systems deviates from most commonly held definitions, in principle, it is not that different from Sun's definition. Particularly not if Microsoft starts licensing source code for NT the way it licenses source code for LAN Manager. While hard-liners will scoff, more pragmatic users, particularly those making significant commitments to Microsoft applications, are likely to embrace this notion of proprietary openness.

Windows Open Services Architecture (WOSA)

At a recent industry conference, some light was shed on just how Microsoft will implement its definition of open systems in the form of the Windows Open Services Architecture (WOSA). WOSA addresses the need to improve the way Windows-based systems tie into larger corporate computing environments. The architecture is the culmination of work underway at Microsoft to address the need to connect Windows-based workstations to heterogeneous environments within the enterprise in a seamless, standard fashion.

The goal of WOSA is to make it easy to connect Windows with enterprise-wide computing environments while hiding the complexity from the developer as well as from the user. Through WOSA, Microsoft wants to be able to provide an open environment for the development and use of Windows-based applications along with access to enterprise information resources.

LAYERED APIs. WOSA is designed to provide a formal isolation layer between Windows-based applications and a variety of connectivity services in order to work with a variety of heterogeneous environments. It will provide ISVs and corporate developers with an open set of APIs to write to in order to access a range of back-end services. An application using these APIs will be able to interoperate with multiple environments concurrently. A facility Microsoft calls the Service Provider Interface (SPI), allows a wide range of service providers to write implementations for their specific environments. By supporting this interface at the back end, any Windows application can access the service without modification.

WOSA SERVICES. The Windows Open Services Architecture includes services for data access, messaging, distributed file and print services, systems management, and host connectivity. Microsoft is working in conjunction with other leading computer industry companies to deliver specific customer solutions under the WOSA umbrella.

EDA/SQL Extends the Reach of ODBC Applications

One of the first announcements of joint development that addresses WOSA connectivity was made with Information Builders, Incorporated (IBI). The two companies will jointly develop a Microsoft Open Database Connectivity (ODBC) driver for IBI's enterprise data access/structured query language (EDA/SQL) client/server software. This driver will allow developers to write applications for Windows and other platforms using ODBC to access all data sources available to EDA/SQL, which currently number more than 50 relational and nonrelational sources, including data managed by IBM's Information Warehouse.

The prerelease Software Developer's Kit for ODBC currently available will be replaced with a final version during the first half of 1992. The ODBC driver for

EDA/SQL will be available in 1992. It will be provided by both Microsoft and IBI at no additional charge as a part of the EDA/Link communications component for Windows and other platforms.

How Real Is WOSA?

Is WOSA a ploy or a central part of the Microsoft strategy? Bill Gates claims that WOSA is real today in the current Windows product, that it already has a strong set of services, and that it will continue to evolve for the corporate customer. The IBI announcement is significant, because even if EDA/SQL doesn't achieve broad market acceptance, it provides both proof of concept and a method for customers to achieve the functional benefits of open systems, although the method is not ideologically pure. With WOSA, Microsoft will be able to capitalize on the discrepancy between the promise of open systems and the availability of sufficient standards and products.

A Sun-like strategy with a PC implementation could give Microsoft just enough credibility with those open systems proponents who prefer to have something they can implement in the near term to something that will take years of debate to achieve. De facto standards have a habit of working their way into formal standards, a fact that is not lost on Microsoft. —*M. Goulde*

FOCUS: STANDARDS

The AlphaWindow Standard: A Solution Looking for a Problem?

When the X Window system was developed, it was in response to a specific challenge facing the MIT Project Athena. That problem was to provide access from various workstations to applications running on heterogeneous systems across a network. Now, the Display Industry Association (DIA), which was created in January 1991, has released a specification for windowing on character-based terminals. The key questions is, "Does this capability address a real problem, or is it a way for makers of character terminals to face competition from X terminals and low-cost PCs?"

Character Terminals Still Prevail

Character terminals remain the most prevalent desktop devices attached to systems of all sizes of Unix systems for one simple reason: They are the cheapest devices available. In fact, 75 percent of today's Unix software packages are character based, and 90 percent of Unix software developers take requirements of character terminals into consideration when developing software. Windowing character applications without change to their source has been possible for some time using software products that run on standard terminals, but these put a load on the host. Terminal vendors have also developed proprietary windowing terminals, but these have been unsuccessful due to the lack of standards for software developers.

DIA Addressed the Need for a Specification

The DIA stepped in to fill the void. It comprises hardware, software, and communications companies with an interest in the character terminal market. This composition reflects the components required, specifically terminal firmware, window management, and host-based expansion cards enabling AlphaWindow terminals to be supported.

The specification provides a standard way for software to interact with and drive terminal firmware so that a windowed environment can be delivered while minimizing the additional load on the host. This functionality can be exploited by providers of window manager software, while applications need no modification. The AlphaWindow emulates a graphical user interface including typical windowing functions, decoration, and mouse support.

The specification also allows developers to exploit the functionality either directly or through a toolkit interface, allowing applications to exploit a mouse, push buttons, dialog boxes, etc.

Components of the AlphaWindows Specification

DISPLAY SERVER. The Display Server resides in the terminal and is responsible for all window-clipping and per-window finite-state maintenance. Although most vendors will implement the Display Server in firmware, there is no reason that it could not be written as a software function for a programmable terminal.

Window Manager. The AlphaWindow manager would normally run on the host, although this is not mandatory. It is the window manager that determines the "feel" of the terminal, enabling it to mimic Motif or OpenLook. There is no reason why the window manager could not be implemented on some intermediate platform, such as an intelligent multiport display controller.

Varying Levels of AlphaWindow Capability

An AlphaWindow terminal can support four groups of function. The first is mandatory, the others optional, allowing terminal vendors to differentiate their products in price and features.

GROUP 1. Group 1 represents entry-level functionality, with support for features such as opening and closing windows, cutting and pasting between windows, creating and deleting borders, changing window size, etc.

GROUP 2. Group 2 defines mouse support, enabling a local mouse to work with the AlphaWindows terminal.

GROUP 3. Group 3 defines window decoration, such as borders, scrollbars, icons, and push buttons, and it enables AlphaWindows to more closely emulate a graphical interface.

GROUP 4. Group 4 defines the communication protocol between the host and the AlphaWindow terminal. The illustration demonstrates the appearance of a Group 3 terminal with the JSB MultiView Mascot's window manager.

AlphaWindow Terminal Characteristics

An AlphaWindow terminal has to be designed specifically to support the specification. It runs a small piece of code called the display server. While display servers may differ in appearance, allowing the vendor to determine the "look," the window manager running on the host determines the "feel," or behavior.

The specification allows vendors to determine the underlying emulation, the number of sessions supported, the keyboard layout, and the type of pointing device provided. A vendor can also offer various screen sizes, resolutions, fonts, and color support to differentiate its products.

AlphaWindows and Software Support

The DIA will provide a specification to software developers enabling them to develop applications to directly exploit the AlphaWindows terminal. Toolkit vendors will be encouraged to interface to the standard through an AlphaWindow interface library.

The specification is built on top of existing terminal characteristics, which means that applications written for specific devices, like the VT220 or Wyse 60, will continue to run on AlphaWindow terminals.

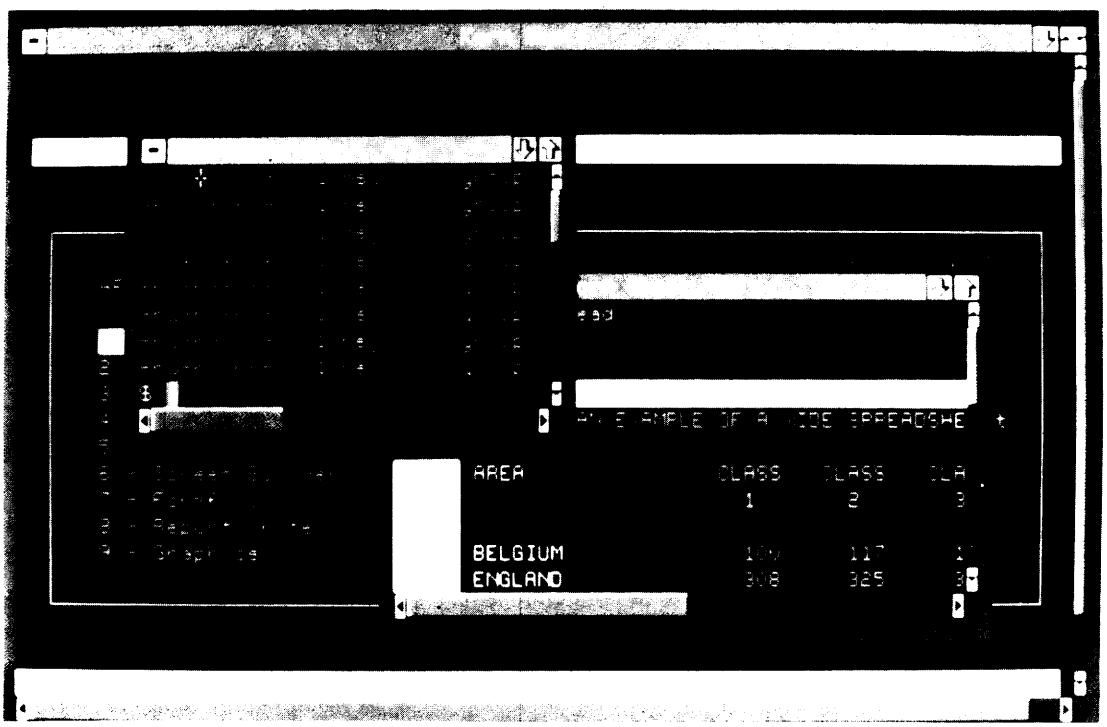

Illustration. Overlapping windows on a Group 4 compliant AlphaWindow color terminal from Microvitec.

A Specification Is Not a Standard

The DIA recognizes that its specification is just that—it is not a standard. It does hope, however, to see its specification adopted as standard if it can achieve widespread support among vendors and users.

The specification will only become a standard if it addresses a need in the marketplace. The list price for a monochrome AlphaWindow terminal will be in the $650-to-$800 range, significantly below that of an X terminal.

Finding Value in AlphaWindows

However, it is not the hardware savings that are likely to create interest in AlphaWindows. The ability to window and run multiple applications concurrently without modification is likely to appeal to cost-conscious companies as a step along the migration path to open distributed systems. The investment made in training users in how to use a windowing system with their familiar applications will be easy to carry forward to newly developed GUI applications.

In addition, as users who have been limited to full-screen character applications become familiar with the characteristics of a windowed environment, they will become valuable participants in the process of designing new GUI applications. The result should be applications that are better suited to the needs of those who will be using them. This may be the ultimate benefit of AlphaWindows to users. *—M. Goulde*

VENDOR FOCUS: SUN MICROSYSTEMS

SunSoft Releases Tools for Transport Independent RPC

Sun's software subsidiary, SunSoft, has announced the availability of the ONC RPC Application Toolkit for developers to use in building distributed applications for Sun's Open Network Computing (ONC) distributed computing environment. (See sidebar for a description of ONC.) The toolkit will provide developers with a higher-level access to the Transport Independent RPC (TI RPC) and give them the ability to more easily create applications that can access multiple network transports, including TCP/IP, OSI, and even NetWare's IPX/SPX.

The toolkit is based on the ONC TI RPC and includes the ONC Edition RPC Tool from NetWise. There are nearly 1.8 million ONC licenses, but relatively few of these licensees have built client/server applications using older RPC development tools. The complexity of writing to varying network interfaces has been one of the constraints in the development of open, client/server applications.

TI RPC Implementation History

The earlier RPC implementation used the sockets interface, but TI RPC uses Transport Layer Interface (TLI) because it is protocol independent, provides a well-defined layered model, has better negotiating primitives, and because TLI is part of SVR4. TI RPC was designed to shield the developer from having to know about specific transports, and the new Application Toolkit further simplifies the development process.

Moving ONC's RPC to TLI was achieved by re-implementing the naming and binding interface to the RPC library in a transport-independent way. The RPC protocol itself was already transport independent and required no change. Applications that already use the RPC library will be compatible with the new TI RPC without modification. Extensions to the original ONC RPC are included in the toolkit to handle synchronous and asynchronous connections, and callback. The toolkit also includes a debugging mode.

Solaris: Foundation for New Technology

SunSoft is using the migration to Solaris 2.0 as an opportunity to upgrade many components of ONC. The full set of ONC services will be included as a part of Solaris 2.0 on both the SPARC and the Intel platform. In addition, ONC source is available for licensing by other vendors and will be implemented on a variety of non-Solaris platforms.

SunSoft will continue to provide the older RPCgen facilities as long as customers are interested, but the superiority of the new toolkit will likely reduce RPCgen's importance.

Toolkit Components

The RPC Application Toolkit includes a source code generator based upon NetWise's RPC Tool technology, and the Transport Independent RPC (TI RPC) technology developed by AT&T and Sun for System V.4. The RPC Toolkit generates C code automatically, which handles client/server communications. It also provides for compiling and testing on the target system.

ONC Services in Solaris 2.0

The TI RPC toolkit will provide sorely needed assistance to developers building distributed application services using ONC services. ONC includes many components, several of which have become de facto standards in the industry. These components include:

Remote Procedure Call (RPC). RPC is a set of operating system-independent operations for executing procedures on remote systems over a network. It provides a logical client/server communications structure that handles low level interprocess communication details transparently.

eXternal Data Representation (XDR). XDR is an architecture-independent method of representing data, resolving differences in data byte ordering and data type size, representation, and alignment. Applications that use XDR may exchange data across heterogeneous hardware systems without having to explicitly make data transformations.

Network File System (NFS). NFS is a distributed file system that provides transparent access to remote file systems on a network. It ensures that files that have been made available to the network appear local to a user's machine. Enhancements to NFS have been made in Solaris 2.0 to improve security by using Kerberos authentication and to improve performance through multithreading support.

Network Information Service Plus (NIS+). NIS+ was introduced with Solaris 2.0. It works with ONC and is a hierarchical enterprise naming service for the management of dynamically changing network environments. It stores system information such as host names, network addresses, and user names. It provides a central point for logging changes to the network, such as adding, removing, or relocating resources. NIS+ is compatible with the existing NIS, which was formerly known as Yellow Pages (YP).

Lock Manager (LM). LM supports file- and record-locking across the network, allowing users and applications to coordinate and control concurrent access to information.

Remote Execution Service (REX). REX is used to execute user commands or programs on remote systems.

NETdisk. Diskless workstations can boot across a network from servers that support the ONC/NETdisk protocols.

Automounter. Automatically mounting and unmounting remote directories on an as-needed basis is a service provided by Automounter. This capability provides both increased transparency and increased availability of NFS file systems. Automounter supports replication of frequently read and infrequently written files, such as system binaries, by allowing the remote mount points to be specified as a set of servers rather than a single server. This places a copy of the file on each server for read purposes.

PC NFS Daemon. The PC NFS Daemon is a small program that runs on an ONC server, off-loading the burden from DOS PCs of having to handle authentication and print-spooling services.

TLI. While not a part of ONC, Transport Layer Interface (TLI) provides a communications layer that makes the RPC protocol independent, allowing RPC programs to run across multiple network transports. The enhanced RPC written to the TLI is known as Transport Independent RPC (TI RPC).

TI RPC in Operation

It is important not to confuse TI RPC with SunSoft's Tooltalk. TI RPC is a network facility that runs a procedure on another system and returns a result to the calling application. Tooltalk operates at another level of abstraction. It allows applications to connect with other applications and exchange data interactively without previous knowledge of one another. Tooltalk, therefore, supports interapplication communications that are more ad hoc and conversational than an RPC.

Functioning at an even higher level of abstraction is the Distributed Object Management Facility (DOMF) that Sun is working on with Hewlett-Packard. The TI RPC operates at an underlying level, providing basic services required by the DOMF, but a programmer using DOMF would never have to use TI RPC directly.

Toolkit and Source Licenses Immediately Available

SunSoft announced immediate availability of the toolkit on Solaris 1.0 It will be available with Solaris 2.0, and applications developed under Solaris 1.0 will be source compatible with the 2.0 implementation. Source licenses are available, and it is anticipated that many ONC licensees will make the toolkit available on their platforms.

Third parties, including Novell, Banyan, Borland, Lotus, Oracle, Sybase, and Informix endorsed the ONC RPC Application Toolkit for developing distributed applications.

Tools like the Application Toolkit are designed to help ensconce the ONC RPC in the installed base as the dominant client/server mechanism before OSF's DCE can take hold. Although the NCS RPC included as a part of the DCE has been available from HP/Apollo, Digital, and other licensees for some time, its real impact won't be felt until DCE is widely installed and used. —*M. Goulde*

The Fourth Annual

Executive UniForum Symposium

Migrating to Integrated Open Systems: Tools, Tactics, and Tradeoffs

May 5 - 7, 1992

Four Seasons Biltmore Resort Hotel Santa Barbara, California

Registration Fee: $1095

Sponsored by

Patricia Seybold's Office Computing Group, UniForum, X/Open

The 1992 Executive UniForum will go beyond basic concepts and abstract issues. It will zero in on the issues and challenges of actually building and delivering business applications that leverage the best that open systems have to offer. Executive UniForum is designed to meet the needs of decision makers and managers responsible for developing and implementing information technology strategies for their organizations.

Updated Conference Agenda

Day 1 Morning General Sessions ▲▲▲

Business Cases and Investment Planning Issues

- The Business Case for Open Systems
- Matching Information Technology to Business Strategy
- Customers' Experience with Open Systems: Do They Deliver?
- Planning an Enterprise Open Systems Architecture

▲ Afternoon Breakout Sessions

Track I: Building Solutions

- Alternatives for Enterprise Development
- How Vendors Implement OLTP Applications on Today's RDBMSs
- How Users are Implementing OLTP Applications on Today's RDBMSs

▲ Afternoon Breakout Sessions

Track II: Management & Planning

- The New EIS—Employee Information Systems
- User Requirements for Open Systems
- *Prolog:* X/Open "Xtra '91 - The Bottom Line"
- Open Systems RDBMS Development

Day 2 Morning General Sessions ▲▲▲

Deployment and Migration Planning Issues

- Managing Distributed Open Systems
- Next Generation Operating Systems
- Cutting Through the Noise: How I Bought an Open System

▲ Afternoon Breakout Sessions

Track I: Building Solutions

- Open Systems Foundations for Workgroup Applications
- Frameworks for Building Client/Server Solutions
- X/Open Workshop: Building XPG3- and XPG4- Compliant Applications

▲ Afternoon Breakout Sessions

Track II: Management & Planning

- Managing Open Systems Development Projects
- Managing Applications Development for Portability
- Planning Security for Distributed Open Systems

Day 3 Morning General Sessions ▲▲▲

Looking Ahead

- Object-Oriented Development: Ready for Prime Time?
- Standards in Computing: Where Are They Going?
- Information Access: Any Place, Any Time, Any Way

For More Information: Fax (617) 742-1028 Call (617) 742-5200 or (800) 826-2424

Patricia Seybold's Computer Industry Reports

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*	6 issues & tapes per year	$395	$407	$419
☐ *Paradigm Shift—Patricia Seybold's Guide to the Information Revolution*	6 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: _____

Fax No.: _____ 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-392

Topics covered in Patricia Seybold's Computer Industry Reports in 1991 & 1992:

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

Office Computing Report

1991—Volume 14

#	Date	Title
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
10	Oct.	Positioning Windows Word Processors—Looking Beyond a Set of Features
11	Nov.	Keyfile—Bringing Imaging and Workflow to the Desktop
12	Dec.	IBM/Lotus Relationship—Building a Platform for Communicating Applications

1992—Volume 15

#	Date	Title
1	Jan.	The Groupware Phenomenon—Does It Focus on the Proper Issues?
2	Feb.	Digital's TeamLinks—A Renewed Focus on the Client Desktop

UNIX in the Office

1991—Volume 6

#	Date	Title
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
10	Oct.	OSF's ANDF—The Key to Shrinkwrapped Software?
11	Nov.	The SQL Standard—Can It Take Us Where We Want to Go?
12	Dec.	Positioning Desktop Options—How Does Unix Fit in the Client Environment?

1992—Volume 7

#	Date	Title
1	Jan.	Downsizing with Open Systems—Can Unix Symmetric Multiprocessing Systems Meet MIS Requirements?
2	Feb.	System V.4 and OSF/1—Matching up in the Marketplace

Network Monitor

1991—Volume 6

#	Date	Title
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
10	Oct.	OSF DME: The Final Selections—OSF Chooses an Object-Oriented Management Platform
11	Nov.	ANSA—A Model for Distributed Computing
12	Dec.	PowerBuilder—Graphical, Client/Server Database Applications Tool

1992—Volume 7

#	Date	Title
1	Jan.	Securing the Distributed Environment—A Question of Trust
2	Feb.	HyperDesk DOMS—A Dynamic Distributed Object Management and Applications Development System