Patricia Seybold Group

Editor-in-Chief
Michael A. Goulde

INSIDE

OPEN INFORMATION SYSTEMS

Guide to Unix and Other Open Systems

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

Integrating Applications in the Real World

Evolution, Not Revolution

By Stanley H. Dolberg

IN BRIEF: Businesses are challenged with selecting next generation architectures and technologies with which they will build tomorrow's distributed applications. Their success or failure in meeting this challenge may determine the competitive fate of the organization. While frameworks, standards, technologies, and products are emerging that will support the development of true open information systems, the risks of early adoption are holding many cautious buyers at bay. Understanding the state of the art and the options available will help move many users from a wait-and-see mode to getting pilot projects off the ground.

Report begins on page 3.

Changing of the Guard

New Leadership at HP and Digital

The processes were as different as the cultures of the two companies. On July 16, Hewlett-Packard announced in a press release that, on October 31, 1992, John Young, age 60, would be succeeded as president and CEO by Lew Platt, an HP executive vice president and head of the Computer Systems Organization. On the same day, the word got to newspaper reporters that Kenneth Olsen had resigned as CEO of Digital Equipment Corporation effective October 1 and that the company's founder would be replaced by Robert Palmer, head of manufacturing and the manager responsible for the Alpha project. Word of Olsen's resignation had reached the outside world after having spread across Digital's corporate network. Digital employees were visibly upset, and it seemed that the entire company was frozen in its tracks.

The challenges faced by Platt and Palmer are as different as the circumstances surrounding their rises to power. Platt merely has to stay the course and make sure that his charges continue to execute a plan that has been in place for number of years. Palmer, on the other hand, will have to scramble to find a plan to rescue Digital from turmoil and confusion. He has to reduce costs without eviscerating Digital's product development, support, or selling functions. He has to find a better way to rationalize a product line built on four processor architectures, a half-dozen operating systems, and a Byzantine array of software applications.

Palmer is in a much better position than Platt to leave his mark on the company. He can make radical changes in organization, products, and directions, and those changes would be accepted as appropriate actions by a new leader. Under Palmer's leadership, we expect Digital to place more focus on Alpha as the strategic platform for Digital's customers and to encourage them to move from the VAX at an accelerated pace. We also expect Digital to emphasize Intel-based machines that aren't PCs. At the risk of angering some customers, MIPS hardware will probably be dropped as soon as Alpha systems can be delivered as replacements. The balancing act between Ultrix and OSF/1 will probably end with OSF/1 emerging as Digital's single Unix offering. Taking a page out of HP's playbook, Palmer might even create two separate but equal hardware lines, one supporting OpenVMS and the other supporting Unix.

Why should he take these actions? First of all, Digital needs to streamline its product line, partly for the sake of its customers, but mostly for the sake of a sales force that is too often criticized for not knowing the company's products well enough. Complex products are hard to sell. Products that don't comprise a rational strategy are impossible to sell. Second, Palmer should eliminate dead-end products because he can. Olsen couldn't disown his children, but Palmer has no such ties.

Platt is a manager in the mold of John Young. While Young was at the controls of HP, he guided the company along with an almost invisible hand. Many people didn't even recognize him at the announcement of HP NewWave in 1987. Platt is also likely to use a light touch.

Olsen, however, ruled with a heavy hand. His spontaneous appearances in the development labs came to be dreaded interruptions. Palmer is known for quick action. Olsen took forever to make decisions.

We will barely notice the change at HP when Platt takes over, but the changes forthcoming at Digital will be explosive. Sometimes things have to get worse before they can get better.

OPEN INFORMATION SYSTEMS

Editor-in-Chief
Michael A. Goulde

MCI:
MGoulde
Internet:
mgoulde@mcimail.com

Patricia Seybold Group
148 State Street, 7th Floor,
Boston, Massachusetts 02109

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

Open Information Systems (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 *Open Information System*, 148 State Street, 7th Floor, Boston, MA 02109.

Integrating Applications in the Real World:

Evolution, Not Revolution

The Real-World IT Architecture Problem

The Challenges Faced by Users

In the real world of commercial information systems, four broad challenges test the ingenuity of users:

- Defining the business requirements that information technology must serve into the future
- Designing the next-generation architecture and applications to meet those business requirements
- Selecting information technologies with which to build that next generation of applications
- Fitting existing applications and technologies into that next-generation architecture

Success or failure in meeting these challenges can determine the competitive fate of an organization or business. The high stakes involved have pushed some users into aggressive technology plays, while others hedge.

Rapid Technology Cycles Detour Integration

Dramatic cycles of technological innovation in hardware, communications, and software engineering in the computer industry have led futurists to describe an era of infinite, inexpensive MIPS, natural person/machine interactions, and unlimited access to data and applications. In these futuristic scenarios, technology enables a seamless relationship between computers and people, work and play, hardware and software, telecommunications and entertainment. But, in the near term, rapid technology cycles actually slow the availability of functionally integrated applications. Short horizons of technology obsolescence discourage users from building integrated environments because it is safer to wait for the dust to settle. A difficult dilemma stares users down: whether to wait for the better future, or to commit to building badly needed applications now.

The Lure of Open Distributed Computing

At one time, all the computer memory in the world measured less than 64KB, and all the computer software in the world ran in that physical address space. From this lean legacy, the commercial computer industry developed around the inflexible, centralized, off-line mainframe which ran under the one program, one user model. One program, one user soon evolved to many programs, many users in the breakthrough era of the online interactive minicomputer.

One problem with minicomputers was that all those application programs had unique user interfaces, file formats, and data structures. Computers from different vendors could not share applications, applications running on the same computer could not share data, users could not easily access host applications or data on other systems, and the per-seat economics of time-sharing systems almost guaranteed compromised performance. The age of personal computing dawned and gave rise to individual prerogative. The tension between

personal computing and integrated computing spawned the current vision of open distributed computing, in which users exercise freedom of choice in determining the personal application environment but still have access to applications and data in an open systems framework.

The confluence of new ideas and technologies for enabling heterogeneous distributed computing has galvanized user expectations that hardware, operating systems, networks, database management systems, development tools, and user interfaces should be selectable, and that designing and implementing an open, flexible, but integrated application environment should be reasonably easy.

The Good News / Bad News of New Models

Low-cost computing platforms and flexible computing models have attracted users to open systems. Promising new paradigms like object orientation and industry-consensus distributed computing technologies have provided powerful incentives for users daring to try to overcome uncertainty and pursue the apparent economic and functional benefits of open distributed computing, often on an enterprise scale.

The good news is that agreement has been reached on fundamental object-oriented distributed computing components within the Object Management Group (OMG), and industry consensus has been declared on Distributed Computing Environment (DCE) and Distributed Management Environment (DME) from the Open Software Foundation (OSF). Additional good news is that these models will facilitate the development and deployment of open distributed applications. The bad news is that, to really play in this world, new applications have to be developed or existing applications have to be fundamentally rebuilt.

Complexity, Uncertainty, and a Dearth of Tools

General agreement on the benefits of distributed computing contrasts sharply with the confusing voices instructing users on how to design and implement distributed systems. Some of the underlying technologies necessary to develop and deploy distributed systems have been available. However, the task of building, managing, and maintaining distributed systems has been too complex to undertake under real-world financial constraints. Uncertain progress in the standardization process for enabling technologies and the lack of development tools has narrowed the group of users interested in distributed computing to risk-taking early adopters. And for the early adopters, what comes next? Early adopters of distributed computing will be likely to develop symbiotic relationships with vendors, acting as an outboard product development/quality assurance function. The early adopters will feed the sorely needed real-world requirements into the Release 2.0 and 3.0 tools that will broaden the market and entice the next wave of more risk-averse application builders to work in the distributed model.

Strategic Plans Collide with Tactical Realities

In the real world, where users and applications meet to do useful work, the seamless future looks attractive. Unfortunately, it's not entirely clear how to get there from here. While vendors relentlessly cycle hardware and software offerings paced to technology evolution, users cycle application software paced to the needs of the organization. Most users have "legacy" systems and applications chugging alongside advanced client/server development projects, illustrating this asynchronous response to rapid technology change. Many users have struggled to develop a strategic information architecture and plan, but few have the means at hand to integrate existing applications and systems into that strategic plan.

The Bottom Line: Users Want to Integrate Existing Applications into the Distributed Model

Application developers working in the distributed model are often asked to integrate existing applications with new paradigms and technologies in order to leverage the value of extensive investments in data and skills, or at least to take a comfortable first step into the unknown. Including existing applications and data sources in an evolving information architecture can constrain the pace and the manner of implementation of new technologies and paradigms. As users grapple with these challenges, the flow of new technologies continues. The economic friction point at which users should integrate existing applications is a moving target.

Interoperability and Application Integration

Users Take Control

In the late 1980s, the computer industry user community substantially wrested control from the vendors. This shift in power derived from a broad set of causes, among them user demands for standards compliance, open systems, and effective means to rationalize and integrate the systems already in place.

Standards Enable Interoperability; Interoperability Enables Integration

Users now understand that standards are the building blocks of interoperability. The ability to interoperate is a necessary but insufficient basis for distributed computing. Standards are the building blocks of interoperability, and interoperability, tools, and distributed computing infrastructures are the building blocks of modern integrated applications.

The drive from users for standardization of system platforms in every dimension, from the OS interface (POSIX) to data access language (ANSI/ISO SQL) to network and system management (OSI, DME), has pushed vendors into providing support for a common body of third-party application software that can be flexibly designed into integrated enterprise information systems.

Levels and Dimensions of Application Integration

The hierarchy of application integration consists of four levels, shown in Illustration 1, extending from base-level consolidation on the desktop to the ideal of dynamic, event-driven, peer-level application relationships in the network.

Application Integration Hierarchy

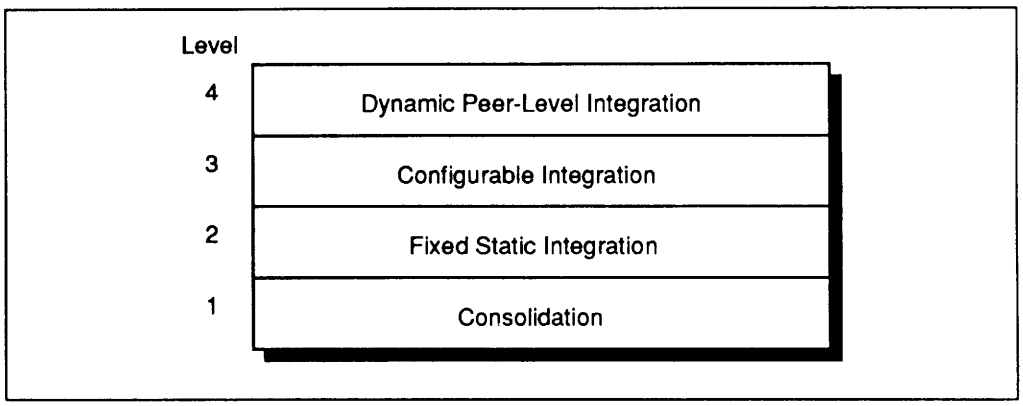


Illustration 1. A hierarchy of application integration beginning with the simplest form, consolidation, and evolving to full peer-to-peer integration.

Consolidation. Consolidation describes the collection of applications "under one roof" for the user, but does not imply functional integration between the applications.

Fixed Static Integration. Fixed static integration describes the minicomputer integrated office systems (IOS) systems, the current generation of PC-based office suites, or the custom integration of multiple applications for a vertical market. Relationships between applications are tightly prescribed and difficult to modify.

Configurable Integration. Configurable integration means that the linkages between the applications can be reconfigured through developer intervention. Configurable integration includes the ability to integrate additional applications as needs change or as application functionality must be extended.

Levels and Dimensions of Application Integration

Dynamic Peer-Level Integration. Dynamic peer-level integration is the ultimate goal. To achieve this, the applications and the relationships between them must be defined with a highly generalized model, such as object orientation. At this level, intelligent transactions occur in the network between applications, transparent to the user, delivering data or services to the user desktop. The relationships between applications are completely event driven, offering the ability to enhance the functionality to the desktop without changes to user application code, yet offering the user the experience of a tightly-woven integrated application environment.

Dimensions of Integration

The level of integration that is achieved depends on how the dimensions of integration, shown in Illustration 2, are managed.

Dimensions of Integration

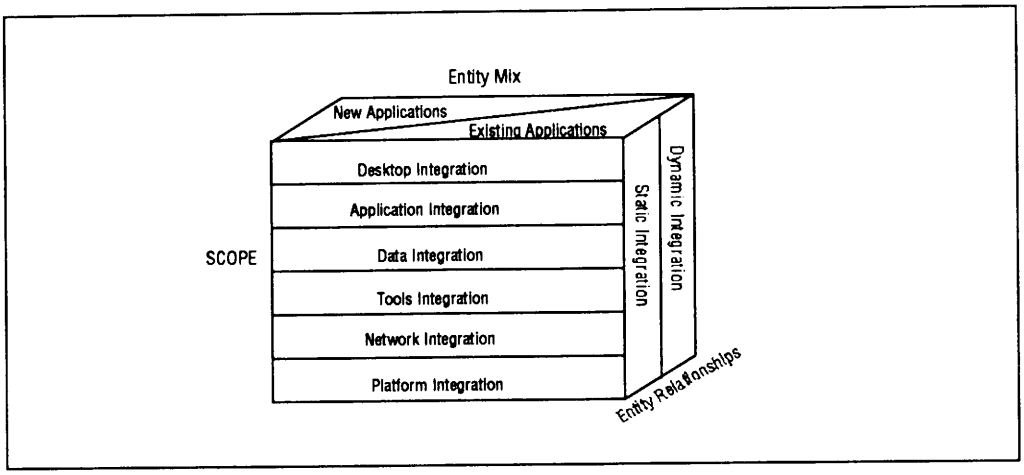


Illustration 2. Integration has three dimensions encompassing scope, entity mix, and entity relationships. The degree of integration achieved depends on how well these three dimensions are managed.

The Dialectic of Integration: Integration, Dis-Integration, Synthesis

Users Seek Virtual Integration: Tightly Wired Functions, No Strings Attached

As personal and corporate computing environments have become increasingly complex, the need to simplify the user view of information systems has grown dramatically. The goals of integration have circled this issue with mission statements such as "offering consistent operating methodologies," "increasing ease-of-use," "improving accessibility of resources," and "simplifying presentation to the user." But integration has generally implied a rigid fastening of component parts in a proprietary fashion—not an attractive proposition in the current climate. Ultimately, users will seek the functional equivalent of tightly wired proprietary integration, but with no strings attached. This motivation drives the dialectic of integration toward synthesis of applications for the user. Unfortunately, the process of achieving application synthesis passes through a painful state of application *dis-integration*, which is where we are today.

Operating Systems Become Software Integration Platforms: Enableware

Meeting the goals for application synthesis will depend substantially on the breadth and depth of vendor system offerings as they evolve from tuned proprietary environments to more loosely integrated platforms of hardware and software from many sources. Bundled system software has grown from an operating system into an increasingly functional software integration platform.

Software integration platforms generally include operating system, network transports, database management, base development tools, system management tools, distributed file

systems, and graphical user interface support. The growth from system software to "enableware" implies explicit support for standard technology components of distributed systems such as distributed file systems, TCP/IP networking, Motif and/or OpenLook graphical user interface presentation, and future support for comprehensive frameworks such as DCE, DME, and distributed object management systems.

The Evolving Concept of Integration

Integrating the computing environment surfaced as an issue with the advent of interactive, multiprogramming minicomputers. Users could run as many programs locally as they could master from the command line, such as word processors, electronic mail, and data extract programs customized for proprietary file management systems and databases. Users could even, with considerable effort, run programs or access data on remote systems. The pieces were all there, but there was no integration in the user application environment.

The Integrated Office System: A Brief History of Static Integration

From this minicomputer cauldron came the first generation of IOSs, such as Data General's CEO and Digital Equipment Corporation's All-In-1. Minicomputer-based IOSs brought several previously unrelated command-line-driven, character-oriented office applications together behind a multilevel menu system, all based on a common host-based filing system. Just as character-based applications can be masked behind a graphical encapsulator, these first-generation, horizontal, integrated applications masked the command line with relatively intuitive character-based menu systems. However, the underlying applications were unrelated, were written in low-level languages, and did not conform to standard text or document formats. But, pulled together with copious amounts of low-level glue code and operating under a common user interface, IOSs defined the benchmark for integration. The user working environment took on sharp, though ultimately limited, definition. Freed from the arcane command line, the average user could create or edit documents, develop spreadsheet models, or send messages or files using electronic mail.

Vertical Integration

Paralleling the advent of integrated horizontal software, the early "best of breed" vertical applications such as general ledger and hospital patient records systems took to the integration road. By integrating other financial-oriented or hospital-record-oriented applications into suites of applications, vendors began to address the user problem associated with incompatible applications running on incompatible computers storing redundant data in incompatible formats about a common set of accounts or group of patients.

Personal Computers and the Era of Dis-Integration

When personal computer software began to offer faster, better-featured applications than the IOSs, the era of dis-integration began. Vendors attempted to redevelop IOSs to operate as the integration mechanism for all the applications in enterprise. But the static relationships among application entities in the IOS were an inherent weakness that kept these systems from adapting to the personal computing phenomenon. After generally unsuccessful attempts to compete head-on with best-of-breed PC applications, IOSs incorporated the PC within the IOS environment through a programmatic precursor of client/server computing.

IOSs evolved to accept personal computer-generated documents, spreadsheets, and data through proprietary integration efforts with the leading personal computer software packages such as Lotus 1-2-3, WordPerfect, and dBase, ultimately leading to back-end servers that provided generalized filing, mailing, printing, and networking services to personal computer clients.

Aster*x from Applix persists in this vein. Aster*x offers a Unix-based, office-oriented, compound document framework with which users can integrate applications using a proprietary scripting language called Extensible Language Facility (ELF). Integration with

personal computer applications is provided by filters that provide bidirectional translation services between Aster*x and de facto PC software formats.

Networked Personal Computers: Beginning the Long Road Back

The local area network (LAN) industry proposed a different framework for integration based on the personal computer as the application engine. Novell, 3Com, and Banyan offer models that base the user's working environment on user-selected personal computer-based applications, while offering centralized services for file storage, backup, printing, electronic mail, and wide-area communications gateways. These models constitute the first generation of PC-based distributed systems.

PC LAN network operating systems (NOSs) integrated the organization in fundamental ways, but their infrastructure has not provided adequate access to business applications. Other than supporting terminal emulation, NOSs by themselves offer little in the way of integrating user environments into the core business information systems. LAN users also confront problems with integrating diverse applications on the desktop and among incompatible desktops. Individual PC software vendors addressed the issue with the equivalent of PC-based IOSs. Starting with the Macintosh clipboard and extending to the Object Linking and Embedding (OLE) model from Microsoft, personal computing environments have steadily evolved toward more flexible user-directed desktop application synthesis.

Integrating Applications the Old-Fashioned Way: Working with Core Technologies

Locking into Component Technology

Table 1 lists and describes many of the core integration-enabling technologies that offer pieces of base capability required to build an integrated, networked system of heterogeneous applications under a graphical user interface. Unfortunately, attempting to build a complex integrated environment with these basic technologies presents some complex problems. Each technology requires programming in a different low-level language or primitive interface. Hand-coded "glue" programs are required at every point where two or more applications need to interoperate. And, perhaps most daunting, working at the lower levels requires that permanent decisions be made at development time about the software "components" in the system, which, once again, sacrifices flexibility at run-time.

Hand-Coding with Low-level Tools

Hand-coding with third-generation C language and remote procedure calls (RPCs), as shown in Illustration 3, allows well-defined, "application-intimate" connections to be programmed between applications. Details of the network protocols and services required to enact the interapplication call are handled by the RPC, but the RPC code "stubs" that establish and manage the interapplication communication are compiled into the client and server application programs, creating a coherent and integrated, albeit inflexible, system.

Working with Higher-Level APIs Is the Better Way

Incorporating the ever-broadening range of de jure and de facto standard technologies and products poses a serious challenge to the battle-hardened professional integrator as well as to the earnest in-house application developer/integrator. The search for a better way has spawned new paradigms, technologies, and products, some of which are radical and some, merely evolutionary. The key attribute of effective frameworks for heterogeneous integration is programming interface abstraction. APIs serve to abstract the libraries that actually handle underlying diversity in network protocols, SQL and non-SQL data access languages, user interface toolkits, and operating system interfaces. This abstraction is essential to building distributed systems without locking either the application or the developer into lower-level components.

Underlying Application Integration Technologies

X Window	The MIT-developed standard for networking of graphical applications. X11 Release 5 has widespread industry support for cross-platform distribution of graphics across networks. X Window consists of a base library of drawing routines (Xlib) and a base toolkit of widgets (Xtk) based on Xlib. (See *Unix in the Office,* April 1992.)
TCP/IP and other Network Protocols	The de facto standard protocol for Unix-based local area networking. TCP/IP is widely supported as the basis for interoperability across Unix and proprietary platforms. LU6.2 from IBM, DECnet, IPX/SPX NetWare protocol, and others are also used.
SQL	ANSI and ISO standard for access to relational database systems. The SQL standard has suffered from incompatible implementations across vendors. Standards bodies continue work on consolidating details and advanced features to leverage the near-consensus on SQL into a basis for reliable interoperability across vendors.
OLE: DDE, DLL	Microsoft's Object Linking and Embedding (OLE) model of integrating applications offers a single-user capability for ad hoc integration of applications operating under Windows and the Macintosh. OS/2 support may follow. OLE uses Microsoft's Dynamic Data Exchange (DDE) protocol to establish ad hoc links between applications. Applications interoperate as client and server through the OLE API that is implemented in Dynamic Link Libraries (DLLs).
X.400	Consultative Committee for International Telephone and Telegraphy (CCITT) standard mail and messaging protocol compliant with ISO/OSI seven-layer communication standard.
X.500	CCITT ISO/OSI-compliant standard messaging protocol that includes user directory capability.
DCE, ONC	Alternative, RPC-based distributed computing development and deployment frameworks. Distributed Computing Environment (DCE) is from OSF, and Open Network Computing (ONC) is from SunSoft. Both include RPC development and run-time environment, distributed file system, naming services, time synchronization, and network security.
DME	OSF Distributed Management Environment, based on DCE, will offer a framework for the development and integration of system and network management applications for distributed computing environments.
Motif/OpenLook	Alternative Unix system graphical window management and look-and-feel systems. Both are built on the X Window System. Motif comes from the Open Software Foundation and offers window management and a style guide. OpenLook is offered by Sun Microsystems and includes some desktop management features.
ORBs	Object Request Brokers (ORBs) are the core building blocks in the object-oriented distributed computing model. ORBs handle requests for service and locate and manage the delivery of requests and the results of requests to and from distributed objects. The Object Management Group (OMG) Common Object Request Broker Architecture (CORBA) defines the base standard for ORB operation.
Product APIs	Application programming interfaces (APIs) offer programmers a documented set of libraries that simplify interactions with the application, e.g. Sybase Open Server.
RPCs	RPCs provide a low-level programming interface with which applications running remotely can call procedures from each other in defined client/server relationships. The programming environment for RPCs is similar to C. With RPCs, applications can be integrated based on specific client/server dependencies that are compiled into the source code of both the server and client applications. RPCs are being enhanced and extended to provide transparency of calling across heterogeneous networks. RPCs play a key enabling role in OSF's DCE and in object-oriented distributed computing frameworks such as DOMF and DOMS.
IPCs	Interprocess communication (IPC) facilities provide mechanisms for processes to share information, typically in the form of messages, semaphores, or shared memory. IPCs can occur locally or remotely through TCP/IP.

Table 1.

RPC Model for Integrating Distributed Applications

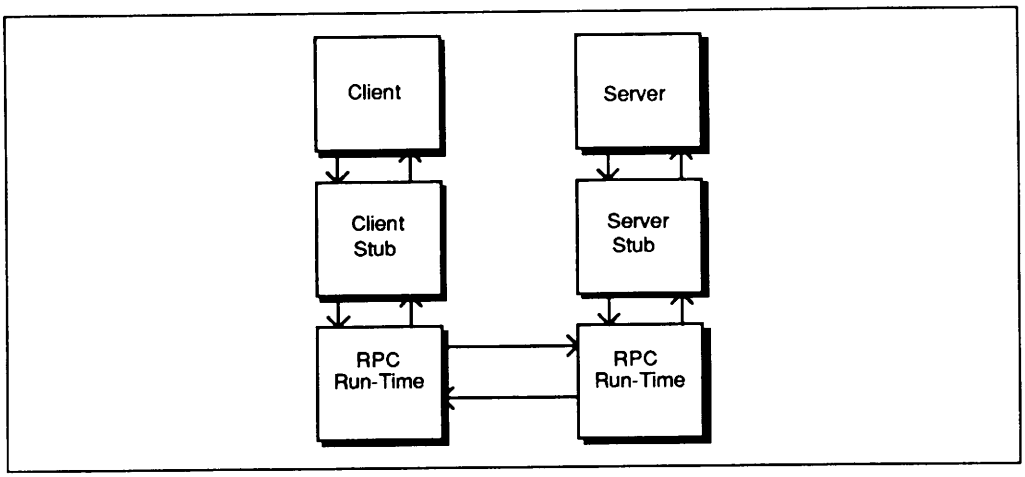


Illustration 3. RPCs alone provide a low-level programming interface and network protocol with which applications can be integrated based on specific client/server dependencies that are compiled into the applications that are integrated with this mechanism. RPCs also play a key enabling role in the broader scheme of OSF's DCE and object-oriented distributed computing frameworks, such as the HyperDesk DOMS.

Niche Tools Make Core Technologies Workable

The availability of comprehensive tools for integrating applications in distributed environments has severely lagged demand. Niche products have flowed into this void to solve pieces of the larger problem. Niche tools build on one or more underlying technologies for integrating distributed systems to provide increased functionality, or at least to provide a higher-level programming environment to facilitate the development process. They can offer a degree of control over the core technologies enabling a developer to address the broad scope of an integration effort by assembling a customized development environment.

GUI Builders Offer Methodology Integration, Cross-Platform Consistency, and Portability

Graphical user interface (GUI) builder systems are widely used to develop the portion of an application with which the user interacts. They address the productivity problems inherent in working with the low-level drawing routines and skeletal widget sets available with the X Window's and Microsoft Window's drawing libraries and toolkits, and the "higher-level" Motif and OpenLook toolkits. GUI builders sometimes add compilers, interpreters, scripting languages, and alternative libraries of drawing routines and higher-order widgets to these core technologies in order to enhance the speed and ease of development for GUI front ends.

GUI builders can enable the integration of applications, at least at an enhanced level of desktop consolidation. If used within a context of specified style conventions, GUI builders can provide common "drivability" across a range of applications which can then be arrayed and accessed on a common desktop environment.

GUI Code Becomes Integral to the Application. Standalone GUI builders generate user interface source code that becomes integrated with application source code through "callbacks" or source code modules that link user interface events (mouse-clicks on a button) with functions in the underlying application. GUI callbacks are coded using C or some other programming methodology within or external to the GUI builder tool and are ultimately compiled into a single executable program. With this development approach, a single executable process performs all interactions with the user, data, and display device. In

order to change the user interface, source code changes must be made with the GUI builder, and the cycle of coding callbacks, recompiling, and relinking the application is repeated.

GUI builders range from tools that build static screens with buttons, sliders, list boxes, and pop-up or pull-down menus, sometimes grouped as interactive design tools (IDTs), to tools that offer features that assist the broader development process, sometimes grouped as user interface management systems (UIMSs). UIMSs offer such features as optional proprietary high-level scripting languages; a range of code generation modes; proprietary "convenience" widget libraries; and flexible, interpreter-based, callback development environments that facilitate testing and debugging the user interface with the live application.

GUI Cross-Platform Portability. Cross-platform portability of applications built with GUI builders determines the ability to deploy the application in a mixed vendor environment and to provide continuity across platforms. Cross-platform portability of GUI-based applications can be ensured through the careful use of standard libraries, such as UIL for OSF/Motif or C, instead of the more comprehensive but proprietary convenience libraries offered by most of the GUI builder products.

GUI Cross-Windowing System Flexibility Helps Integrate for the User. Cross-windowing portability can be as important as cross-platform portability, but for different reasons. Consistency across windowing systems can provide an integrated user view of the environment, providing consolidation of desktop applications. While popular GUI builders such as UIM/X from Visual Edge (Quebec, Canada) and Builder Xcessory from Integrated Computer Solutions (Cambridge, Massachusetts) concentrate on building Motif-based user interfaces, other GUI builder systems and more comprehensive development environments such as Uniface from Uniface Corporation (Alameda, California) and Galaxy from Visix (Reston, Virginia) support cross-windowing deployment. Portable GUI builder systems, such as Neuron Data Open Interface from Neuron Data, Incorporated (Palo Alto, California) and XVT from XVT Software (Boulder, Colorado) have abstracted the assortment of graphical user interface programming interfaces (OSF/Motif, OpenLook, Microsoft Windows, Presentation Manager, and Macintosh) to a common API. The product API maps to the native libraries for each windowing system and look-and-feel at the time of compilation, supporting the cross-platform delivery of a single development effort. Illustration 4 outlines Open Interface's architecture.

Cross-Windowing Issues: Display, Conversion, Graphical Encapsulation

When the goal of base-level application consolidation requires the integration of existing applications, the ability to display and to work with non-native applications in the desktop of choice becomes a critical success factor. VisionWare Limited (Leeds, England) offers products that support cross-windowing display, conversion, and graphical encapsulation of Unix-based applications into the MS Windows world. One such product, XVision, supports the display of the OSF/Motif or OpenLook host-controlled X client in an MS Windows environment. Optionally, XVision can use MS Windows as a window manager to control local MS Windows applications and remote X Window clients. XVision allows data to be cut and pasted between the local MS Windows applications and the X-based applications, and DOS and Unix applications can also be hot-linked through the MS Windows DDE facility. VisionWare's PC-Connect product supports the display and manipulation of character-based Unix applications within the MS Windows environment, also supporting dynamic connections between applications. VisionWare also recently announced plans to partner with Alex Technologies (London, England) to deliver a product called Alex for Windows that will support the delivery of host-based character applications on the MS Windows desktop running completely in a Windows-compliant mode.

Neuron Data Open Interface's Architecture

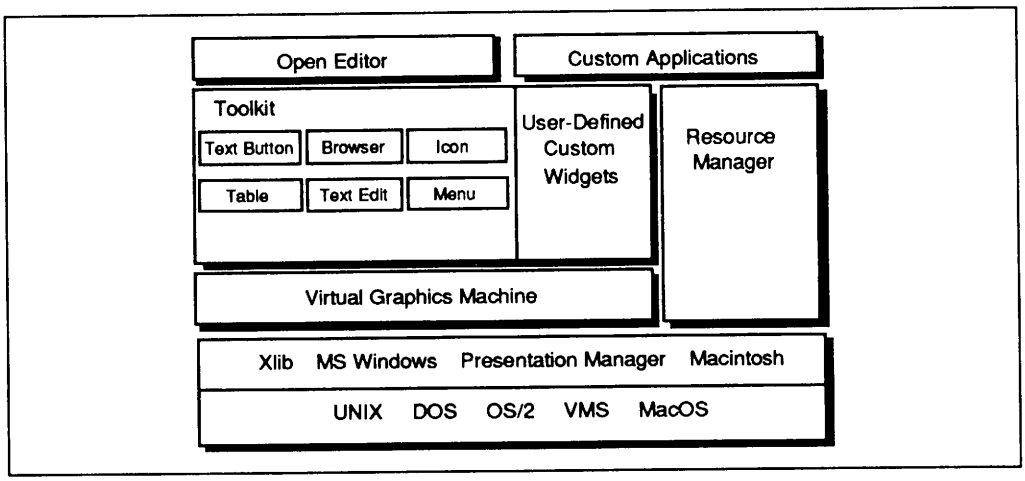


Illustration 4. Portable GUI builder systems, such as Neuron Data's Open Interface, have abstracted the assortment of graphical user interface programming interfaces (OSF/Motif, OpenLook, Microsoft Windows, Presentation Manager, and Macintosh) to a common API that maps the product-unique API libraries to the native libraries for each windowing system and look-and-feel at compile time, supporting cross-platform delivery. This system requires support for the development environment on each target platform.

Another approach that helps to consolidate applications on the desktop is taken by DeskTerm from IXI (Cambridge, England). DeskTerm includes a DeskTerm protocol that translates character sequences to graphical, windowed events. DeskTerm offers two ways to operate Unix-based character applications within a standard windowing system—either by modifying the source code to directly access the DeskTerm protocol or leaving the application unchanged and using the DeskTerm SoftOption scripting language to describe the character sequences to the DeskTerm protocol. DeskTerm supports delivery of the Unix character application with either an X Window-based OSF/Motif style presentation or an MS Windows 3.0 presentation, helping the user consolidate applications under a common environment and providing a base-level of integration.

PC Desktop Integration: OLE

Desktop integration in the PC world means two things: the fully integrated Macintosh environment and the Windows Object Linking and Embedding (OLE) API. In its current form, the OLE model of integrating applications offers a single-user-oriented capability for integrating applications operating under Windows and the Mac OS, and, perhaps in the future, OS/2. Object-linking supports the creation of "hot links" between applications that incorporate the same data to ensure consistency of that data if it is changed, and object embedding allows the user to create and edit compound documents with the experience of working from a single integrated application.

OLE operates on top of Microsoft's Dynamic Data Exchange protocol (DDE) to provide integration on the Microsoft Windows desktop. OLE uses the DDE protocol to establish ad hoc links between applications. Applications interoperate as clients and servers through the OLE API that is implemented through a client and a server Dynamic Link Library (DLL). With OLE, objects (data) from "foreign" applications can be embedded in unrelated applications that cannot edit the embedded objects. Selecting the embedded object actually launches the underlying application to enable the editing and updating of the embedded object. The recently finalized OLE 2.0 specification from Microsoft will support editing in place. Alternately, the two applications can be linked so that changes in the data file of the OLE server application result in corresponding changes in the OLE client data file. OLE is shown in Illustration 5.

Desktop Integration with OLE

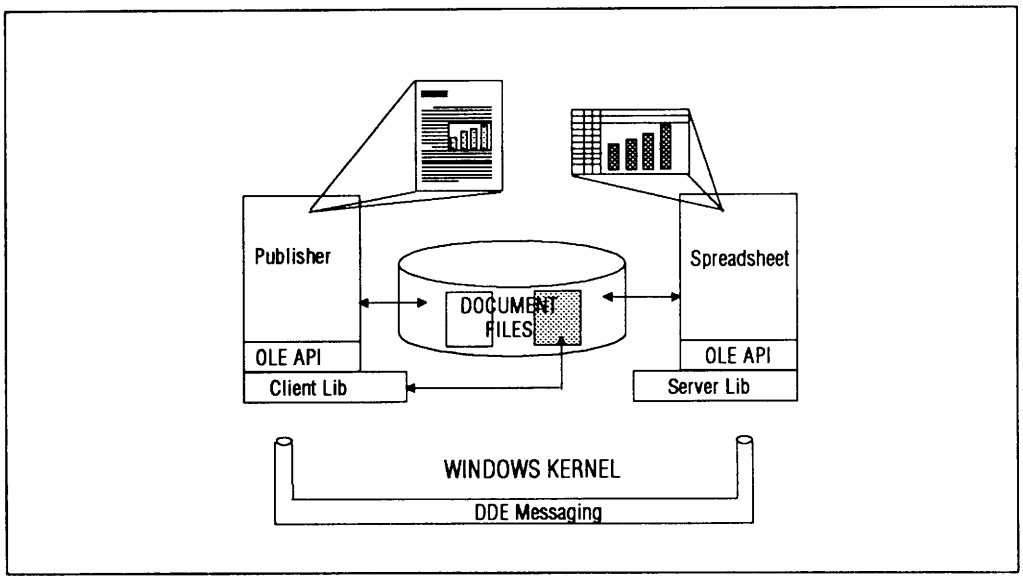


Illustration 5. The Object Linking and Embedding (OLE) model of integrating applications offers a single-user capability for ad hoc integration of applications operating under Windows and the Mac. OLE uses the DDE protocol to establish ad hoc links between applications or to embed objects from those applications in a document. Applications interoperate as client and server through the OLE API that is implemented through client and server DLLs.

Desktop Management Systems Offer Baseline Application Integration: Consolidating Applications and Resources

The base level of integration for the user is the simple consolidation of multiple applications on the desktop under a common look-and-feel provided by the windowing system. Desktop management systems run as applications on top of industry-standard windowing managers, offering the user an intuitive metaphor for the computer system as a set of personal office resources such as desk, drawers, files, file folders, and office accessories and equipment. Desktop management systems offer a simplified view of the operating environment, the available resources, and the actions that can be performed on those resources, such as drag-and-drop manipulation of files for printing, faxing, mailing, or launching.

Desktop Management Systems Can Deploy across Look-and-Feel. Desktop management systems in the Unix environment build on the X Window technology. But on top of X Window, the desktop management systems vary from full commitment to one look-and-feel, such as the X.Desktop system from IXI Limited, which specializes in OSF/Motif, to the abstracted approach of the Looking Glass system from Visix Software, which can switch its desktop presentation dynamically between OSF/Motif and OpenLook.

Desktops Can Launch Custom Integration Applications. Desktop management systems can also serve as the context for user access to iconic representations of hand-built cross-application integration programs that involve remote data or applications. For example, an application developer supporting a marketing department might write an application with embedded SQL statements that uses variables inserted by the user on a form to access sales information stored in a database about certain products over specified time periods. The icon that represents the application can be selected and used to launch the application in the same way that other applications are manipulated. However, today's desktop management systems do not provide a robust development environment for building these applications. Certain vendor-unique desktop management systems, such as OpenWindows from SunSoft (Mountain View, California), are tightly integrated with a more complete set of tools for

building integrated networked applications that exploit the specific platform and desktop management system.

Messaging Systems: Simplifying Interapplication Communication

Message-passing interprocess communication offers a generalized, flexible mechanism through which distributed applications can interoperate in a variety of ways. Messaging systems offer practical means for programming networked interapplication communication by addressing the problems of finding the location of target applications on the network, performing the network protocol handshakes required to navigate the network, and managing the sending and receiving of messages between applications.

Messaging Systems Shield the Developer from Network Details. Messaging systems abstract a proprietary API to a set of C function libraries. From this interface, the application developer can, depending on the messaging system model, be shielded not only from the underlying network protocols but also from having to know where the called processes are located, or even with which processes the communication occurs. Message Express from Horizon Strategies (Needham, Massachusetts), Softbench Broadcast Message Server from Hewlett-Packard, and the SunSoft ToolTalk component of the OpenWindows development environment all offer messaging system functionality.

Messaging Connects the Parties: Conversation Is the Key to Integration. Messaging systems provide directory services, some of which are object oriented, that provide efficient ways to locate target applications and data. Messages can be used to transact anything from SQL statements to a request for analysis of values submitted to a simulation program. Messaging systems effectively address one important dimension of the application integration puzzle—transparent interapplication communication across networks. But messaging systems by themselves are not a development framework designed to build integrated user applications. Like the telephone network, messaging systems connect the parties but do not help them negotiate the contract.

Messaging API Requires New Applications or Encapsulation. Messaging systems require that the applications be newly developed or redeveloped with the product-specific messaging API in order for the application to offer an interface that can interoperate over the messaging system. As one example, Illustration 6 shows how the ToolTalk API works as a messaging system. The HP Broadcast Message Server, recently unbundled from its Softbench CASE product, offers a specific paradigm for encapsulating existing applications with the Encapsulator Description Language (EDL) or, in a future release, with C and C++.

Encapsulating existing applications offers programmatic access either to the full set or to a subset of the application's functions. To date, messaging systems are currently used mostly by ISVs seeking to integrate a set of proprietary CASE or CAD tools into a distributed framework.

Data Integration Tools

Data integration tools, such as Powerbuilder from Powersoft (Burlington, Massachusetts), hide the incompatibilities among SQL implementations and support the development of client/server applications. These applications can call database server resources without specific knowledge of the database SQL syntax or the networking protocols required to make the connections. However, data integration tools do not provide a general methodology for integrating diverse applications, and they operate in the messy world of inconsistent data types, SQL implementations, client/server protocols, error codes, data dictionaries, and distribution methods. They are valuable because they form-fit an interface to the SQL DBMSs that presents a simplified, consistent API to the applications developer.

ToolTalk Message Flow

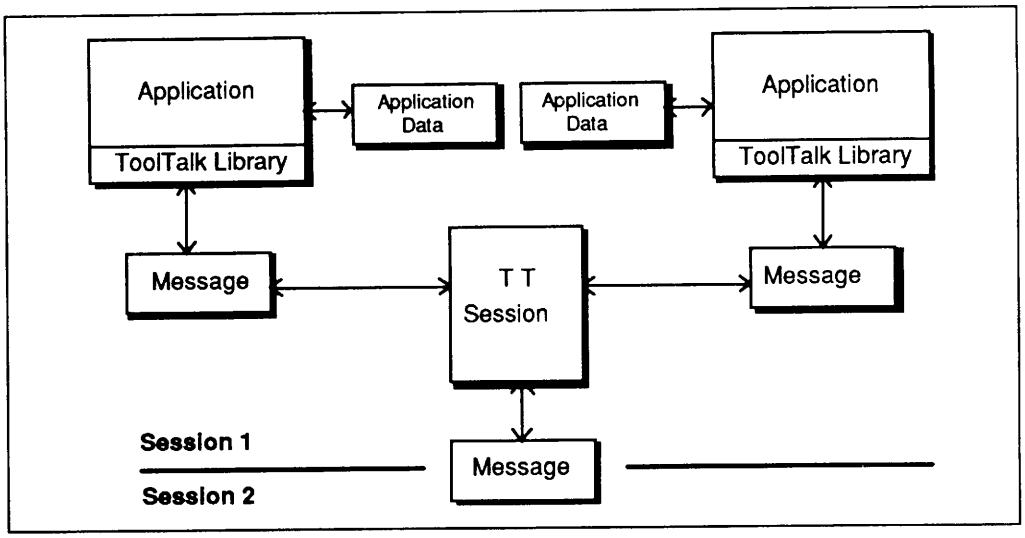


Illustration 6. To interoperate on a messaging system, applications must present a programmatic interface based on the messaging API. ToolTalk either distributes broadcast messages to applications based on the registered requirements of the applications or handles requests for service by matching the request attributes to the capabilities of registered applications.

Integration Development Tools Issues

Niche tools solve some of the problems of building integrated, distributed systems but leave unanswered a variety of questions and issues that need to be considered when working either with them or with more comprehensive frameworks. Developing distributed applications will challenge users for some time to come. However, implementing distributed applications that deliver integrated functionality will pose additional difficulty. Developers today should be targeting at least the level of application integration we have described as *configurable* integration. Issues that have to be considered include whether the tool being used is a high- or low-level tool, the maturity of the technology, the extensibility of the integration architecture, and whether development is dependent on specific tools or technologies. The ideal integration framework would offer programming abstractions across the entire potential scope of application integration, unlimited extensibility, and open interfaces for incorporating third-party tools as they become available. Unfortunately, there are real-world issues to consider in differentiating among niche tools and more comprehensive frameworks.

High-Level Code-Free vs. Low-Level Code-Intensive Development

Integrating diverse applications requires managed connections between applications, across operating environments, and across networks. "Code-free" programming environments offer the highest level of simplification and abstraction for developers integrating applications. Visual programming tools, sometimes labeled fifth-generation languages (5GLs), range in functionality from GUI builders with extensive list box selections to true object-oriented environments that offer flow-charting techniques for building integrated applications from properly designed objects.

While it is a great benefit to the application developer to have powerful tools that hide layers of complexity, it's virtually impossible in the real world to develop a complete application through a completely code-free process without at least some lower-level programming. For example, lower-level programming is required to hook GUI callbacks into underlying applications. To integrate existing applications into object-oriented

environments, lower-level programming is used to build the encapsulation code that mediates between the application command syntax and the interface to the object broker. While 4GLs can call 3GL routines directly, code-free object-oriented tools incorporate 3GL routines as encapsulated objects, requiring at least some C++ or C-based "glue code."

Mature Technologies, Emerging Technologies: Keeping Options Open

Users want to reduce their application development backlogs, but they also want those applications to be developed while platforms, operating systems, networks, and databases are changing. Working at lower levels of programming to integrate applications with mature open technologies yields low risk, low productivity, and most likely low flexibility. Working in emerging technologies implies "bleeding edge" experiences usually associated with research labs. The trade-off between working in lower-level programming environments with standardized technologies and working in emerging higher-level environments with proprietary technologies begs some basic questions about the interplay of applications and organizational change. There is a middle ground where proprietary higher-level application programming interfaces overlay mature technologies.

Scalability: Is There Life after Prototyping?

Prototyping is a critical part of the development process for graphical, networked applications. Rapid prototyping tools support the constructive involvement of users before large investments in system design and programming have been made. Prototyping tools typically include graphical screen design tools, along with a high-level language and interpreter to allow incremental development of the interface and sample application. In some cases, tools have been designed that support not only rapid prototyping but also the generation of a scalable, production-quality application. Crossroad from Crossroad Systems (Boston, Massachusetts), Sammi from Kinesix Software (Houston, Texas), and SuperNova from Four Seasons Software (Edison, New Jersey) offer development environments that support rapid prototyping that flows relatively seamlessly into the implementation of production quality applications.

Extensibility: Integrating the Unexpected

Murphy's law states clearly that no matter how coherent the plan, the unexpected will occur. For example, at one point in time, four applications will be identified that must be integrated for the stock trader's floor, but, six months later, two or three additional applications will appear out of nowhere which have to be included. The extensibility of the integrated environment will depend on how generalized a model is employed to plug together the component pieces.

The Developer's Dilemma: Platform Dependence vs. Tool Dependence

Application developers want to build portable applications, complete projects on time, and bring them in under budget. Working with generic low-level programming tools offers a high degree of application source code portability but sacrifices programmer productivity. In order to achieve independence from system vendor platforms, developers have shifted from 4GL tools tied to proprietary relational database management system (RDBMS) systems to 4GL tools that are independent of RDBMS vendors. Different types of independence are important:

- *Hardware and operating system independence* for applications built with 4GL systems depends on the 4GL vendor porting to a broad array of workstation and server platforms. The 4GL-based application can be compiled and run on any platform supported by the 4GL vendor.

- *Network protocol independence* can be achieved through use of RPCs or messaging systems that offer a common API for access to diverse networks, at least as diverse as those supported by the product. The Uniface and SuperNova 4GL environments, for example, offer network independence by including client/server facilities that hide the underlying transport required to reach a server resource.

- *Interface independence* allows applications to be integrated for the user in the user's environment, which is a critical element in effective integration. Neuron Data offers

interface independence by mapping its common API at compile-time to low-level drawing routines native to each look-and-feel, for example Xlib for OSF Motif. The Visix Looking Glass desktop management system and Galaxy application development environment provide run-time selectable look-and-feel by mapping the common user interface API to the characteristics of the selected look-and-feel under a real-time user interface management facility.

- *Database independence* has been elusive because of the slow progress in the definition and adoption of the SQL2 proposal. However, products such as Uniface, PowerBuilder, and SuperNova provide application development environments that support the building of data-intensive applications without requiring specific knowledge of the data source by the developer or by the application.

Frameworks for Integrating Applications with Open Systems Technology

Working through the Frameworks: Beyond Core and Niche Technologies

Lying beyond the proposition of accomplishing integration by working with core technologies or niche technologies is the world of integration frameworks. Comprehensive integration frameworks build on enabling core technologies and niche technologies to support efficient development of complex, heterogeneous applications.

Although no single development framework covers all of the bases, comprehensive tool frameworks are evolving that will meet the need for the highest levels of integration. A number of development frameworks and tools are available that offer a variety of architectures, development methodologies, and underlying technologies that support the development and deployment of distributed, integrated applications.

Integrating Applications with the Distributed Computing Environment (DCE)

The OSF's DCE supports the development and deployment of large-scale client/server applications. DCE assembles a standard platform of enableware, namely time synchronization services, authentication and security services, distributed file system services, and remote procedure call run-times (see Illustration 7) that the developer can assume will be found on platforms supporting DCE when applications need them to cooperate.

DCE RPC Stubs Are Compiled into Clients and Servers. DCE does not include a full application development environment, but it does include development tools that support the distribution of applications using the DCE RPC. The DCE RPC development tools consist of the C-like IDL, used to create the RPC client and server interfaces, and the IDL compiler, which generates header files, and object code client and server stubs for each remote procedure call interface. Header files and stubs must be compiled into the server and client application source code to enable distributed applications under DCE. The component applications in a DCE-integrated application environment are developed and their interactions configured by the developer at compile time. The DCE structure is shown in Illustration 7.

Applications must be newly developed or fundamentally rebuilt to be distributed and integrated using DCE. Existing applications that cannot be rebuilt or were not explicitly designed to be distributed with DCE can still be incorporated as servers behind hand-built programs that "talk DCE RPC Runtime" to the requester and act as users of the existing database or application. In order for DCE-based applications to operate across different platforms, each system vendor must support the new technology in DCE implemented on its platform in a consistent fashion.

DCE RPC Runtime Library Structure

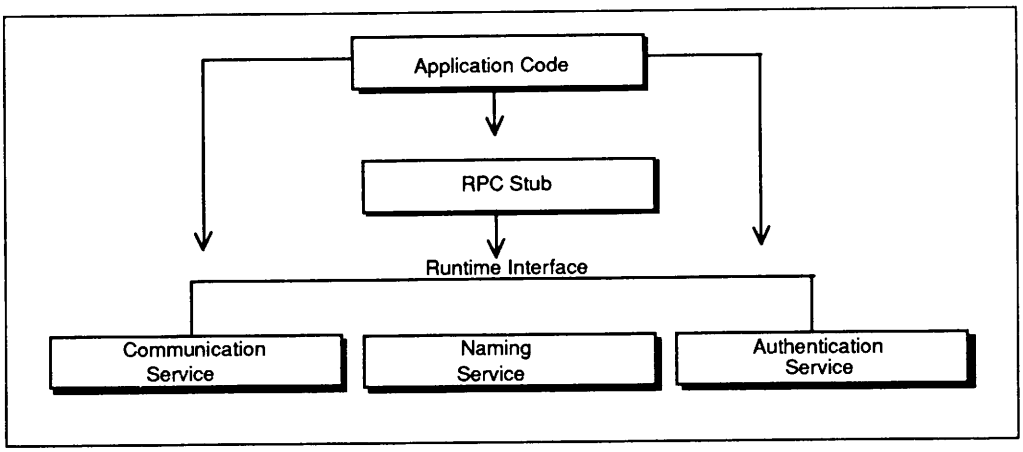


Illustration 7. The DEC RPC Runtime architecture illustrates the integral nature of the RPC to the compiled client and server application code. DCE represents a comprehensive standards-based environment for building and implementing distributed systems that operate as an integrated environment. Implementing DCE requires that new applications be built or existing applications be extensively modified to enable deployment within this distributed model.

Integration with the Distributed Object Model

"...its soul, its whatness leaps to us from the vestment of its appearance... the object achieves its epiphany." —James Joyce

Complex Standards Must Be Set. The phenomenon described by James Joyce cuts to the essence of the distributed object vision shown in Illustration 8. Under the distributed object model, application integration becomes completely run-time user-configurable from the available applications and data sources in the network. The distributed object model enables this ideal state because all objects in this framework, including users, applications, devices, and networks are self-describing, well-behaved, cleanly interfaced, and accessible through secure and well-managed methodology. The distributed object ideal state describes an abstract, generalized model that could remove all obstacles to seamless, user-transparent, real-time, integrated computing. However, before users experience the epiphany of application synthesis, many complex standards must be set.

CORBA Is Step One. The Common Object Request Broker Architecture (CORBA) defined by the OMG represents a key foundation for the framework of standards that must be established for the realization of the object-oriented vision. Product implementations (or instantiations in OO-speak) of the distributed object model include the HD-DOMS from HyperDesk (Westboro, Massachusetts), the Distributed Object Management Facility (DOMF) from Sun Microsystems and Hewlett-Packard, and Application Control Architecture Services (ACAS) from Digital Equipment Corporation.

Integrating Applications in the Object Model with HyperDesk

The HyperDesk HD-DOMS offers a distributed object-oriented framework for developing and implementing distributed applications. As a primary participant in the OMG CORBA specification process, HyperDesk currently conforms its HD-DOMS ORB to the OMG CORBA specification, except for the Static Invocation Interface, which will be supported later in 1992.

HD-DOMS Weaves Object Orientation with Distributed Computing. HD-DOMS combines object concepts with distributed computing concepts to produce a comprehensive environment for integrating users, applications, devices, networks, and data into a flexible, configurable fabric.

Distributed Object Vision

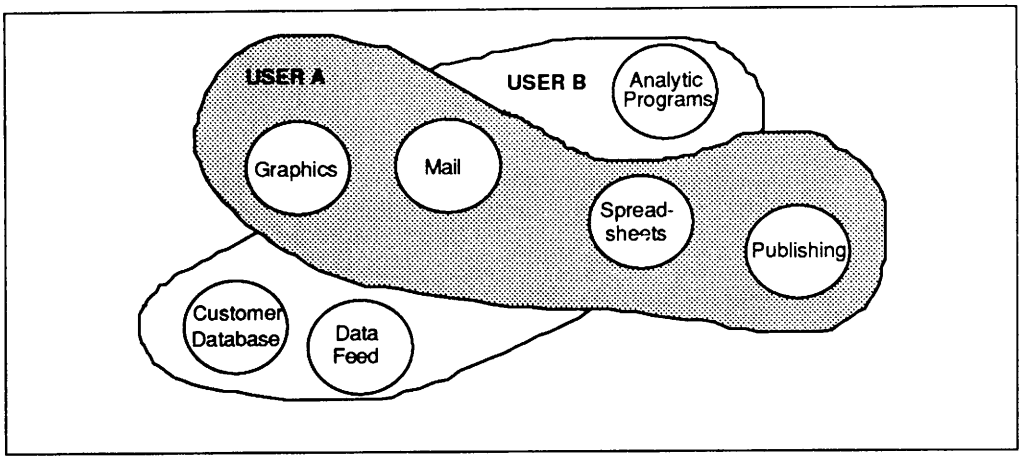


Illustration 8. The distributed object-oriented model describes a state where all applications and data sources in the network exist as granular modules (objects) that are self-contained, self-describing resources that can be configured to operate as an integrated working environment tailored to the needs of the user.

Objects Facilitate Building a Distributed, Integrated Environment. Under HD-DOMS, all applications and data function as objects, interoperating with other objects through the services of local or centralized ORBs. Client and server applications are written largely outside of the HD-DOMS framework, except for the programming of the object interface. The HD-DOMS API-conformant object interface to the application is written using the Implementation and Interface Definition Language (IIDL). With IIDL, developers define *object interfaces* and *object implementations*. An object interface consists of information about the relationships of the object to other objects, the actions the object can perform, and the attributes of the object. The object implementations describe the different ways the object can occur. Object interfaces and object implementations are installed and stored in the HD-DOMS Interface Repository and Implementation Repository in the object database. HD-DOMS insulates the developer from the details of diverse network, operating system, data management, and user interface issues.

When objects (applications) issue requests, the requests are mediated by the ORB, which applies services such as naming and authentication to the requests, ensuring that messages are passed between the correct parties, as shown in Illustration 9.

Existing Applications Can Be Incorporated with HD-DOMS. As with DCE, applications must be newly developed or substantially rebuilt to be fully integrated in the HD-DOMS model. Existing applications can be incorporated unchanged into a HD-DOMS framework as partial participants through a process called *encapsulation*. Developers encapsulate an application by writing a program in C or a shell scripting language on top of which the HD-DOMS client API is layered. Encapsulated applications can be called to perform operations by other client applications with HD-DOMS, but they cannot call on other applications themselves. Source code modifications to existing applications are required using any language that can call C programs to achieve what HyperDesk terms a *foundation upgrade*, which allows existing applications to be integrated more fully into the HD-DOMS user environment.

Synchroworks from Oberon

Visual Programming + Distributed Objects. The Synchroworks product from Oberon Software (Cambridge, Massachusetts) is currently in late Beta-test and is expected to ship in fall 1992. Synchroworks incorporates an object-oriented architecture with a visual programming paradigm. Synchroworks offers the user the ability to configure the user environment from the available resources in the object database through a visual program editor. The Synchroworks visual programming methodology uses building blocks that

represent interfaces to applications, input and output objects such as report or dialog boxes, and devices such as printers and fax machines. Application building blocks are supplied with Synchroworks, created as user interface objects within the visual editor or the class editor, or are built with C++ by the developer and are accessed from the Synchroworks object database during application development. (See *Open Information Systems,* June 1992, for more information on Synchroworks.)

HyperDesk Application Integration Architecture

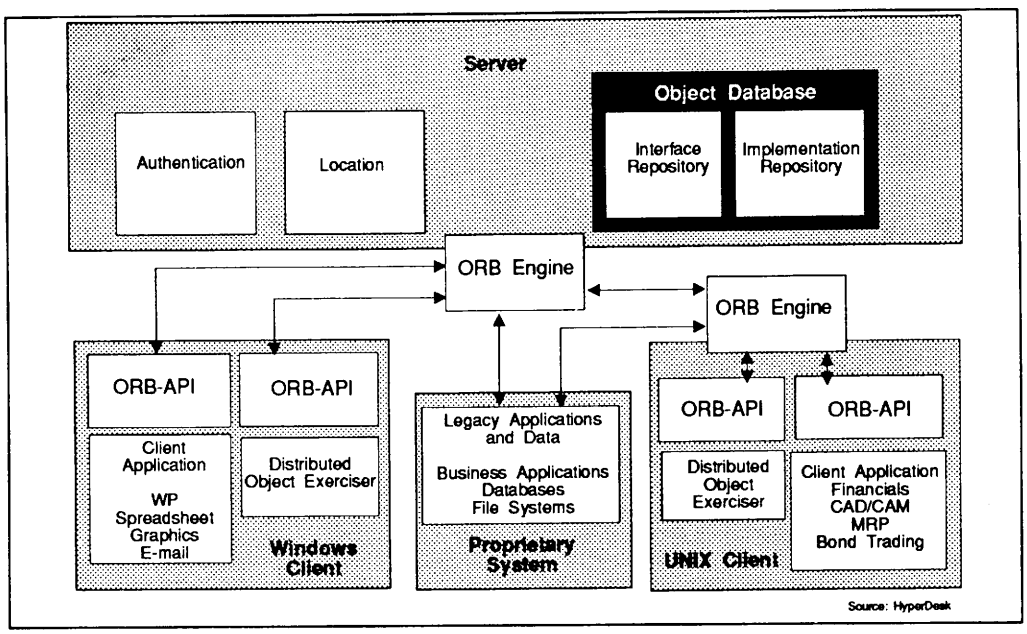


Illustration 9. The HyperDesk HD-DOMS architecture diagram illustrates how existing applications integrate with new applications in an object-oriented distributed computing framework. In what HyperDesk terms "Level 1" integration, unmodified existing applications are first "wrapped" with a program written in high-level scripting language that maps the input/output environment of the application to a simple scripted shell. The application can then be registered as a resource in the interface repository and the implementation repository. Unless the existing application source code is modified to be able to make ORB calls, the application can only service requests from other clients in the framework.

Crossroad Systems Application Integration Framework

Supporting Rapid Prototyping and Production Development

The Crossroad application integration framework from Crossroad Systems is an application development environment that provides support for rapid prototyping of peer-level networked interactions between applications. The Crossroad high-level programming tools target application developers who need to build integrated application environments consisting of new, existing, and third-party applications or data sources. Crossroad offers a general purpose integration framework based on Agent programs that are written to interface to applications by using the application's native functionality. Agents communicate with each other on the network through message-oriented interprocess communication. The Crossroad C-based message-passing architecture is generalized to use a TCP/IP IPC mechanism to ensure the location transparency of Agents running locally or remotely.

Sammi: A Graphical User Environment

User Interface Code Is Isolated from Back-End Applications. Sammi from Kinesix Corporation provides an environment for the networked integration of applications under a

Motif- and X Window-based front end. The Sammi architecture isolates the development and operation of user interface code from application code and databases.

Sammi Runtime Manages Flow Between Clients and Servers. Applications and sources of data are integrated within this framework through the Sammi Runtime Environment, which mediates between the user interface and the back-end applications and data sources that are written or modified to conform to the Sammi API. The Sammi Runtime Environment launches RPCs, on either an event-driven or a polled basis, which connect the user transparently to the applications and data as required by the client system. The Sammi Runtime Environment peer-to-peer configuration can also service back-end applications in real-time environments, or allow the server application to drive the display, as in a training system. (See Illustration 10.)

Sammi Application Integration Architecture

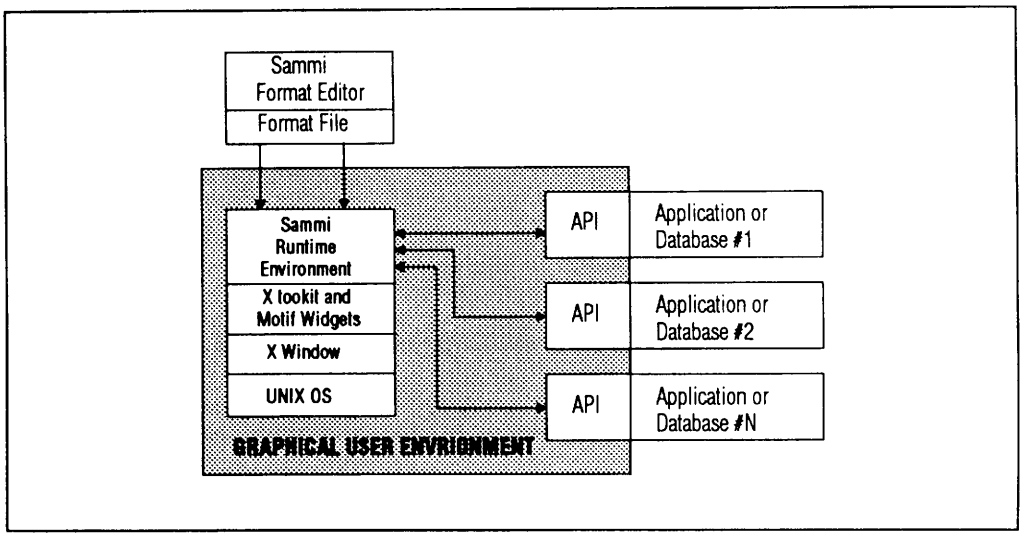


Illustration 10. The Sammi architecture decouples the user interface code from the underlying application and data sources, and insulates the developer from the details of the network protocols through the use of the Sun ONC RPC. Modifications to the user interface do not require code changes in the server applications.

Rapid Prototyping Supported with Dummy Data Links. Sammi supports interactive development allowing the user interface to execute interpretively in the Sammi Runtime Environment without real data links. After the behavior of the user interface has been tested and debugged, the dummy data links can be replaced with references, called Format Keys, to the actual logical servers in the network. Logical servers are applications and databases that conform to the Sammi API and may reside anywhere on the network. The Sammi Runtime can process compound requests that require the division of a logical server request into multiple RPCs targeting multiple and different types of applications and data sources.

The Sammi development environment separates the development of the user interface from the development of the applications accessed over the network. Existing applications must be front-ended with hand-coded programs that know how to pass arguments to the application and get data in and out, and also use the Sammi API.

Galaxy Application Environment

Supports Run-Time User Selection of Look and Feel. Galaxy, from Visix Software, is an object-oriented application development environment for building portable, distributed applications that are integrated for the user under run-time selectable windowing and look-and-feel systems. Galaxy represents a multi-year development commitment by Visix to deliver a comprehensive development environment for portable applications. Early versions

of the Galaxy product were used to develop the Looking Glass line of desktop management products.

Distributed Application Services Incorporates External Applications. The Galaxy API offers powerful abstractions for interfacing to operating systems, networks, and windowing systems. The Galaxy Distributed Application Services (DAS) Server is a multithreaded, RPC-based, interapplication communication mechanism that supports the integration of applications running locally or over the network. The services offered by Galaxy-built applications or non-Galaxy applications can be registered with the DAS Server and can be accessed dynamically by other applications on an event-driven basis. Non-Galaxy applications can be integrated into a Galaxy environment through the addition of hand-coded programs that front-end the application to implement the Galaxy API. Users can access hypertext-based help for all applications integrated using Galaxy through the extensible Help Server.

Built on Abstracted Interfaces to System Resources. The Galaxy architecture offers generalized mechanisms for handling dozens of functions both within and between applications, ranging from the File System Manager to the Cursor Manager. Galaxy includes a robust GUI builder that supports delivery of the application in a choice of look-and-feel including OSF/Motif, OpenLook, and Common User Access (CUA) for Windows or Presentation Manager.

RDBMS-Independent Fourth-Generation Languages (4GLS)

Fourth-generation languages (4GLs) designed for open distributed systems development can provide much of the high-level application development environment needed to build integrated, distributed applications that can even include existing applications.

Uniface from Uniface Corporation

The Uniface 4GL, for example, provides a forms-based application development methodology that insulates the developer from the details of specific user interfaces, network protocols, and data access language uniquenesses of RDBMSs and file management systems. (See *Unix in the Office,* July 1991, for more details on Uniface.)

Three-Schema Architecture Provides Database Independence. With Uniface, DBMS independence is achieved through the use of the Uniface Information and Design Facility (IDF) development environment, which implements the ANSI/ISO-compliant, three-schema architecture, for the development of database-intensive applications (conceptual schema, external schema, and internal schema). This architecture separates the definition of the logical relationships of data and other objects (conceptual schema) in the application from the way particular databases manage the data (internal schema) and from the way the application interacts with the user (external schema).

Conceptual Schema Simplify and Control the Development Process. Once the application conceptual schema are defined, the developer works within a forms-based environment to create the external schema, which automatically build on the logical relationships defined in the conceptual schema. This architecture, combined with the Uniface Polyserver comunications manager, supports the operational integration of multiple databases and multiple applications while insulating the developer from the details of networking, databases, and user interfaces.

GUI Support in Latest Release. The latest release of Uniface supports the development of GUI front ends for applications. Developers writing in the Uniface 4GL can produce Windows, Motif, Presentation Manager, or OpenLook applications from a common code base. Developers write to the Universal Presentation Interface included as a part of Release 5.2 and deploy on any platform for which Uniface supplies a driver. The developer has the choice of sacrificing portability and using GUI-specific features, or staying within common features and writing a portable application. Character terminals are supported with the same code.

SuperNova from Four Seasons Software

Object-Oriented Development Concepts. The SuperNova 4GL from Four Seasons Software offers developers independence across the RDBMSs, GUIs, and networks supported by SuperNova. The SuperNova development environment consists of a relational model data dictionary, a 4GL editor for creating the application logic, and the window editor for creating the user interface. The SuperNova model builds applications as objects along with the functions performed on those objects. Applications do not run as compiled executables, but instead are run as meta code that is interpreted at run-time by the SuperNova engine.

Distributed Processing Supported through the Distributed SuperNova Engine. SuperNova supports distributed processing through the distribution of the run-time SuperNova engine to multiple network nodes. When an application calls a function that is not running locally, the engine refers to the local dispatch table configured by the developer, which contains location information for network resources. The local SuperNova engine then packages the request through its network interface to communicate with the proper remote SuperNova engine, and then returns the results to the requester application. The remote resource could be an existing application that has been modified to interface to the SuperNova engine or an application or database that has been front-ended with a hand-coded C or high-level scripted program.

Cross-GUI Deployment Supported at Run-Time. GUI independence operates at run-time. Developers work with a particular windowing system and look-and-feel during development, but the user can choose to work with the application in a different mode when starting an application. SuperNova currently supports Windows 3.0 and OSF/Motif.

Rapid Prototyping Supported with Referential Integrity. SuperNova supports rapid prototyping through the development of the application with flat files as the data source, which can then be replaced with live links to multiple RDBMSs and other data sources through a "data type" parameter change.

Wizdom Framework from Tivoli: Networked Systems Management and Control

Network and systems management for distributed systems represents a special instance of the challenge of application integration. Users have developed or acquired many applications to manage distributed application environments. Common management applications address the automation of software distribution and installation, user administration, printer administration, and backup and restore, among others. The Wizdom framework from Tivoli Systems offers an object-oriented framework for the integration and operation of systems management applications. Wizdom has been adopted as a core element in the Distributed Management Environment (DME) from OSF, the Atlas architecture from Unix International, and by numerous system vendors in the open systems segment. Developers received the first release of Wizdom in May 1992.

Special Case of Distributed Object Management System. The Wizdom framework is essentially a special case of a distributed object management system. The Wizdom Application Development Environment (ADE) includes the documentation of the Wizdom API, the libraries, and fully distributed debugging tools.

Integrating Existing Applications under Wizdom. While there are few existing commercial management applications for distributed systems, early users of Wizdom have integrated existing user-built management applications for managing relatively small numbers of systems under the common interface. Integration of existing programs is accomplished by encapsulating the existing programs with a Wizdom API-conformant interface. At one site, the administrator of a four-vendor mix of hundreds of Unix workstations, PCs, and back-end file servers used the Wizdom framework to consolidate dozens of shell scripts developed to automate various tasks across the network. Due to the special purpose nature of Wizdom, even existing applications can effectively leverage the efficiencies of operating under the common management interface.

Crossroad Systems Application Integration Framework

Leveraging Existing Management Applications. Interoperability of heterogeneous management applications develops naturally under Wizdom due to the inheritance feature that operates when Wizdom objects are subclassed to build new applications. For example, a printing service can be distributed to and managed for classes of users and classes of systems automatically by using the core object services for object naming and location. Illustration 11 shows the Wizdom architecture.

Wizdom Architecture

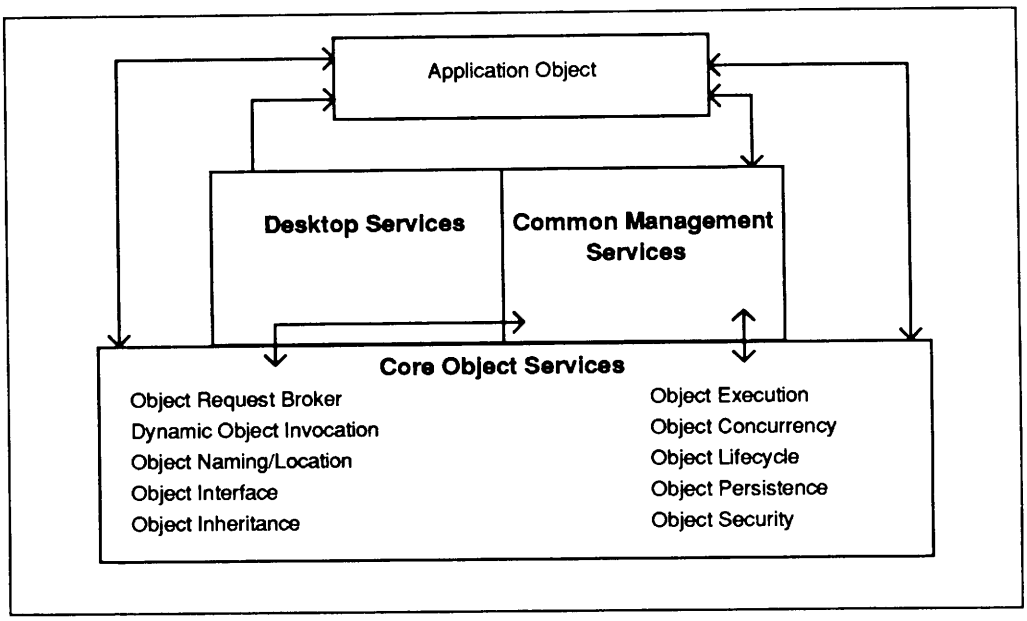


Illustration 11. The Wizdom framework provides an object-oriented framework specifically designed for the integration of distributed systems and network management applications. As with other advanced frameworks, existing applications can be incorporated to some degree into this framework, but the main focus for Wizdom is on the standardization of presentation, use, and interoperation of management and control applications for distributed environments.

Security in Networked Applications

Hitting a Moving Target

Security is a relative term and a moving target. The phrase "secure distributed systems" remains a contradiction in some sectors. Increasing concern over virus contamination and hacker intrusion keeps raising the bar on security. The MIT Project Athena developed the Kerberos model for authentication, verification of data integrity, encryption, and access control over the network. Kerberos has become a basic element in network security schemes included in the OSF DCE, the HyperDesk HD-DOMS framework, the Wizdom framework from Tivoli, and even Microsoft's new Windows NT operating system.

There are other approaches that work within the more tactical application integration frameworks. For example, the Crossroad application integration systems from Crossroad Systems rely for security on the mechanisms inherent in the applications and the application platforms with which Crossroad Agents interact. Crossroad Agents actually export the security functions of the applications they "represent" in receiving and passing messages to other Agents in the network. In another vein, SuperNova provides for access controls enforced by its run-time engine.

Summary and Conclusions

The arrival of broad-scale distributed computing is near. Development tools and frameworks to support the designers, programmers, and implementors have begun to take shape. The movement to the distributed model will be evolutionary, and, for some time to come, will include legacy applications. Development environments that support this evolution have entered the market, pulled by user willingness to trade dependence on innovative tools for highly functional, integrated application environments. The distributed object model provides the most promising paradigm for the future, but the extended time frame for delivery of the standardized distributed object infrastructure will create a large opportunity for less revolutionary approaches to putting the pieces together for the user.

The key to long-term success in these endeavors is making certain that "distributed enterprise computing" does not appear to the user as "dis-integrated enterprise computing."

Next month's *Open Information Systems* will address
Microsoft Windows NT Operating System.

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

Open Systems: Analysis, Issues, & Opinions

TOOL FOCUS: NEURON DATA

Will Open Interface 2.0 Nullify the GUI Wars?

Development of the Virtual Toolkit Approach

The promise of graphical user interfaces (GUIs) for Unix was to give novice users all the power of Unix but to hide the complexities. Unfortunately, when developers encountered the voluminous libraries of the X Window System and its minimal—at best—interface templates, the graphical revolution went into slow motion. Developing X-based programs was a grueling process. When standardized look-and-feel specifications and common widget libraries, such as OSF/Motif and OpenLook, arrived on the scene, the situation improved somewhat. But twiddling C source code to design and edit graphical objects has been a handicap for X Window programming compared to the ease of writing to the more abstract interfaces of the Macintosh and Microsoft Windows. Developers began to address this problem, and GUI-builder programs appeared in the X Window community, some of which rolled over as publicly available products. In these environments, programmers could paint the display layout of a GUI on the screen, have the C code generated for them, and then plug in regular software subroutines underneath. X Window development productivity improved markedly with the availability of these tools. (See "Integrating Applications in the Real World," this issue.)

However, the market for GUI applications has remained fragmented, not only between the X Window look-and-feel camps, OSF/Motif and OpenLook, but also between them and the much larger Macintosh and Windows markets. Neuron Data (Palo Alto, California) was one of the few vendors that recognized that GUI portability among all these platforms could be both possible and profitable, for itself as well as for developers. This was the impetus behind Open Interface, a Neuron Data product that allows developers to design graphical interfaces which run unmodified on OSF/Motif, Microsoft Windows 3.x, Apple Macintosh, OpenLook, and OS/2 Presentation Manager platforms. (See "GUI Portability at Last," *Unix in the Office* Vol. 6, No. 3, March 1991, which covered the first release of Open Interface.) This product helped get developers past the toil of GUI programming and the turmoil of choosing a GUI platform and onto the business of delivering GUI applications to users.

Amplifying the "Kitchen-Sink" Approach

SUPPORTING *ALL* FUNCTIONS ON *ALL* PLATFORMS. When you look at the various GUI platforms, it quickly becomes apparent not only that they differ in appearance but also that each has many unique functions. OpenLook has its pushpin, Apple has its menu bar, Microsoft Windows has its "Task List" dialogue, and so forth. Even functions that look similar are often implemented in very different fashions, such as rubber-band vs. sticky drop-down menus. To build a virtual GUI toolkit supporting all of the GUI specs is a pretty complicated affair. Other cross-platform GUI builders have implemented a subset of the universe of widgets comprising those found across all supported GUIs. But, instead of this approach, Open Interface has implemented a toolkit that is a functional *superset* of all the widget collections. That is, *every* function of *every* GUI look-and-feel is supported on *all* Open Interface platforms. The resulting set of GUI widgets is very large, potentially overwhelming developers, but Neuron Data has opted for richness in order to have the same virtual toolkit on all supported environments without sacrificing functionality.

The Open Interface widget set is not simply a concatenation of all the others. It is a consolidation in which corresponding widgets—such as different pull-down menus—are merged into one representation. Open Interface implements each of the application-level widgets with distinct presentations for each of the target look-and-feel specs. All of these implementations are available on all target platforms and can be selected and changed on all platforms even while the application is running. Neuron Data often demonstrates this by running an application under Windows with Macintosh look-and-feel and on a Macintosh with a Windows look-and-feel. Many skeptics might dismiss such capabilities as frivolous, but they would be surprised, as were we, to hear of two large corporate users of Open Interface that were running the same look-and-feel—in

one case Motif, in the other Presentation Manager—on all their platforms, including Macintosh.

OBJECT-ORIENTED STRUCTURE HELPS INTEGRATION. Good GUIs have a simplicity that masks their tremendous complexity at the code level. The intricacy and interrelationships of pull-down menus, text regions, and scrollbars can be an organizational nightmare. The X Window System and most GUI builders address the difficulty with a form of object-oriented programming, such as sharing type definitions among widget sets and packaging widgets within other widgets. Neuron Data goes further than most in supporting this development paradigm. The first step is encapsulating the data and functions into objects. Open Interface generates fully structured C code and uses C "typedef" definitions, header files, and macro definitions to enforce widget encapsulation. This prevents hand-written implementation software from corrupting the widgets' internal data. Open Interface also supports inheritance, but it doesn't allow a widget to inherit capabilities from more than one parent widget type—that is, it doesn't have multiple inheritance, but it does have basic polymorphism. By using the inheritance mechanism, the developer can build custom widgets that share common characteristics and can thus be integrated more easily into an application framework.

USING EXISTING TEXT-BASED SOURCE MANAGEMENT TOOLS. All GUI components, such as message files and resource files, that are developed under Open Interface are stored as text, not just C source code. Neuron Data uses ANSI C as its target language, since it is most directly portable and widely implemented. Unfortunately, the C language is not completely amenable to object orientation, which gives rise to the limitations in Neuron Data's object approach. When C++ standards are finalized, we expect Neuron Data to move quickly in that direction. The message files and other GUI resources that are stored in text form can be managed under a version control facility like SCCS or RCS along with the C source code. For use at run-time, Open Interface includes a resource compiler that builds resource files into a more efficiently-used binary format.

INTERNATIONALIZED TEXT CAPABILITY. In most software, the text labels, questions, and messages seen on the screen are fixed. The developer has programmed the messages directly into the source code. Consequently, it is difficult to make the software usable for those who speak other languages. Neuron Data addresses this problem by supporting the definition of message libraries. All text that an application presents to the user can be compiled into a file. Using a special filter program, the developer or integrator can generate different versions of the file with messages in different languages. While it might be preferable to have an "internationalization editor," allowing GUI entry of foreign-language messages right next to their developer-language versions, Open Interface's approach is adequate. The internationalization mechanism supports non-roman character sets like Cyrillic and includes multi-byte sets like Japanese Kanji. Unfortunately, text can only be written left-to-right today, which excludes the right-to-left Semitic languages like Hebrew and Arabic. Apart from this, however, Open Interface's internationalization is comprehensive and serves multilingual application projects admirably.

API ALLOWS EXTENSIONS, PORTABLE WIDGETS ... While support for standard widgets is necessary for GUI programming, it is not sufficient. Specific applications—such as cash flows, network management, or customer information—should have distinct widgets as a part of the GUI. Ordinarily, the application developer would have to implement these capabilities using the drawing primitives of the native windowing system on each target platform—largely bypassing the user interface toolkit. This represents a severe compromise to application portability. To address the need for widget extension, Neuron Data has developed a portability layer—implemented as an API—which is identical on all targets. This layer, called the Virtual Graphics Machine (VGM), consists primarily of drawing primitives, plus some windowing functions.

Understandably, the VGM is not a superset of all platform-specific drawing tools. However, it does allow the developer to build existing GUI widgets into new types of widgets. To build the examples mentioned above, a bar-graph widget could illustrate cash flows, a line-drawing widget with some sort of routing indicators could show network topology, and customer information could show maintenance schedules along a time line. The payoff of this approach is near-complete GUI portability. Not only will Open Interface support the developer in creating new widgets, but also the new widgets are inherently portable across all supported platforms.

...BUT REQUIRES A SPECIAL RUN-TIME. Open Interface applications run on a special run-time library. Neuron Data's software is considerably more optimized than standard Motif and OpenLook widget libraries. Almost all Open Interface widgets are actually "gadgets"—to use the X terminology—because they are not built using the window primitives of X. Neuron Data took this route to cut out the high overhead of X Window primitives, and it has added programming workarounds to flesh out standard widget functionality. The result is performance that matches and often surpasses that of

the standard widget libraries. Applications running under X, moreover, are smaller than native applications, and library sizes are smaller.

The downside of Open Interface's powerful run-time is that it adds more layers to GUI software and supersedes standard GUI implementations. An application can behave differently from standard look-and-feel specifications, which is particularly bothersome when it runs side-by-side with native applications. Worse, it must be licensed for each machine on which it is run, adding to the cost. And while X hostings of Open Interface applications tend to be smaller and faster than ordinary Motif and OpenLook applications, the wholesale replacement of look-and-feel code makes applications running on the Macintosh larger than native applications.

Neuron Data packages its run-time library in shared-library form whenever the platform supports shared libraries. This includes most Unix hostings, VMS, Windows 3.x, and OS/2. This approach reduces application size on these particular platforms. While shared libraries are often a hassle to administer from the standpoint of shared memory parameters and file configuration, they can significantly improve memory and disk space management by reducing application size.

Enhancements in Open Interface 2.0

NEW WIDGETS AMPLIFY GUI COMPONENTS. Neuron Data has taken some of the initiative in developing GUI functionality beyond what is provided in the standard GUI toolkits. These basic widget sets include primitive interface components, such as buttons and text fields, as well as higher-level components, such as pull-down menus and scrolling lists. Open Interface 2.0 gives developers additional functionality in grouping and organizing existing widgets. Version 2.0 includes an enhancement of its "List Box" widgets, which implement various forms of row-column organization of data and can be used to build spreadsheet programs. The enhancements allow the display attributes of a List Box cell or range of cells to be changed from the attributes of the rest of the List Box. The List Box also allows more sophisticated input handling; about all it doesn't do now is clone the 1-2-3 macro language for you. A new Scrollable Panel extends a panel widget beyond the dimensions of its enclosing window. The extended area is accessed by using scroll bars. This is one of those "why didn't someone do that before?" capabilities. A "Choice Box" widget set allows selection of an item from a list—but not just a list of text strings. The Choice Box can allow selection of icons for things like drawing tools or text attributes. It's a natural for implementing tool palettes. Other widgets from Open Interface 1.0 have been enhanced in a similar fashion.

UPDATES TRACK SUPPORTED PLATFORMS. Since the first release of Open Interface, some of its supported platforms have had software upgrades. Open Interface 2.0 includes enhancements to adapt to the release of Microsoft Windows 3.1 and IBM OS/2 Version 2.0. Windows support now includes Win32 and TrueType font-rendering. Open Interface can also build software for 32-bit-mode OS/2 operation. Unfortunately, it has been scooped by the recent release of OSF/Motif 1.2; customers needing Motif 1.2 compliance will need to wait for the next release or a software patch. Tear-off menus and drag-and-drop will have to wait until the next release of Open Interface.

POSTSCRIPT TOOLS GIVE WYSIWYG PRINTING TO UNIX APPS. High-resolution printing of text and graphics was the enabling technology for desktop publishing, but the Unix community is largely left out. The PostScript imaging mechanism pioneered on Macintosh and embraced by DOS and Windows has not been standardized under Unix, which has resulted in few PostScript-aware Unix applications. Part of the difficulty is the fundamental difference between X Window imaging, which is bit-mapped, and PostScript, which is a higher-level abstraction. Under PostScript, if you draw a one-inch by one-inch square, it comes out that way on any printer. But under X, if you draw a 100-pixel by 100-pixel box, it can be displayed in practically any size and shape. Open Interface 2.0 supports a PostScript printing capability that is transparent to the application. By changing a switch, VGM rendering commands that would ordinarily drive a screen display are changed to generate a PostScript file. This relieves developers of having to generate their own PostScript printer files.

OPEN EDITOR NOW EDITS CUSTOM WIDGETS. Under Open Interface 1.0, developers could create new widgets but could not edit or adapt them with the tool's Open Editor. Since sizing and customization are part of the way GUI builders earn their keep, this inability to modify custom widgets was a serious deficiency in the product. Open Interface 2.0 has augmented the Open Editor so that custom widgets can be placed, sized, reshaped, colored, and otherwise modified just like those supplied by Neuron Data.

STRENGTHENED CODE MANAGEMENT. No matter how powerful the GUI builder, it won't write your entire application for you. Open Interface, like its competitors, still requires coding the application logic in C. All of these tools generate stub code into which the developer

writes the implementation of the various functions. Once the stub has been fleshed out into a full source file, other GUI builders can't handle it any more. And if the developer has to change the GUI, the stub code must be regenerated and source text from the prior implementation must be pasted in and adapted. Open Interface 2.0 keeps these program texts in the development loop. GUI alterations are propagated to implementation code without damaging the handwritten portions of software. This eases maintenance of GUI definitions and implementation code. Added to its textual representation of GUI resources, these features give Open Interface superior project management capabilities.

Cheaper on the Desktop, Too. One nontechnical change is a significant reduction in run-time license fees. Run-times have been lowered from $250 to $95 for the PC and Mac, and from $500 to $190 on other systems. We feel that this makes Open Interface a much more realistic choice, especially for developers. The previous pricing structure simply couldn't be handled by ISVs trying to sell shrinkwrapped applications, even with bulk or bundling licenses. This reduction in run-time fees makes it feasible to use Open Interface for commercially sold applications.

Who's Committing to Open Interface?

NUMEROUS LICENSES—MOSTLY LARGE END-USER ORGANIZATIONS. Neuron Data has piled up the impressive figure of 100,000 run-times since its first release in June 1991. Most of these licenses have gone to large corporations for in-house application development. Neuron Data's customer list reads like the Fortune 1000, which is, in fact, a primary target market for Open Interface. For these companies, multiple desktop platforms are a fact of life, and Open Interface promises to bring some unity and order without compromising functionality. Unfortunately, a lot of the business in that market boils down to client/server database applications running on PCs and using some sort of mainframe data storage. However, several solid graphical database application builders have already staked out this market with GUI builder products. We suspect Open Interface will find limits to penetration except where there is already a significant number of Unix workstations. Indeed, customers cited by Neuron Data are largely in the engineering or financial industries, which do have significant Unix workstation penetration.

WHERE ARE THE APPLICATIONS? While Open Interface promises a brave new world, it will be off-the-shelf applications that deliver it. Such applications, however, remain rare, and Neuron Data can't give out the names of unannounced products or their developers. A product like Open Interface should really be paying off in several applications already. But where are they? In our view, many of the ISVs have largely gone ahead with their own GUI efforts, which may not be compatible with Open Interface. Of course, the aforementioned qualms about portability and the perennially cash-starved nature of the Unix software market might also explain vendors shying away. But what we suspect is that a not-invented-here attitude prevails. Developers may simply not want to be dependent on another software company.

SOME ISVS AT WORK. Unfortunately, participation of independent software vendors in the Open Interface revolution is rather sparse. Neuron Data's biggest success so far is ARC-VIEW, from Environmental Systems Research Institute (ESRI) in Redlands, California. ESRI is one of the leading providers of Geographical Information Systems, and it believes Open Interface helped ARC-VIEW get to market much faster than it would have otherwise. American Management Systems is also developing products with Open Interface. Two aren't exactly a flood, but these are good, solid companies. Open Interface is moving well in Japan, however, a key payoff of its multi-language, multi-alphabet internationalization.

INTEGRATORS: KEY TO BROADER MARKET PENETRATION? The capabilities of Open Interface were really seized by system integrators. Customers like Anderson Consulting and EDS constitute a testimonial to Open Interface. Consultants and integrators don't take the time for setting corporate policies or for building from scratch. To have custom applications come out identically on multiple platforms with a short development time may catch the interest of large purchasers and developers more than a laundry list of features. The upscale integrators are, in our view, the acid test of Open Interface.

Conclusions

Fills Some Important Gaps in X

Like Consistency. One of the serious defects in the common X Window programming model is that it is an amalgam of components operating at different levels. Even using a GUI builder, the developer may have to code to the look-and-feel toolkit, the basic X Window toolkit, the Xlib interface—even to socket-level functions—in the same application. In contrast, the Open Interface toolkit is completely self-contained. There are few real reasons, if any, to go outside the toolkit. Open Interface provides the comprehensive and

consistent interface that none of the Unix/X Window vendors provide.

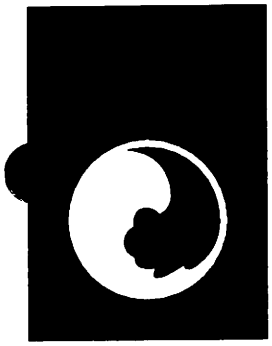
And Profitability. Another minor benefit Open Interface brings to X Window is the possibility that ISVs will make decent money. If a vendor can deliver on Windows and Macintosh, it could earn enough money to support on-going development that would pay off on X-based systems as well. The Windows/Macintosh market is as much as two orders of magnitude larger than the Unix desktop market. Open Interface allows Unix vendors to bring their products into this considerably more lucrative arena. We hope they have enough breadth of vision to make use of it. Or is it just pragmatism they require?

POWERFUL DEVELOPMENT AND ORGANIZATIONAL FRAMEWORK. The principal lessons that Open Interface and other GUI builders embody are that programs are built from more than source code and that each component should be built with an appropriate tool. The dreary chore, inevitable in early X Window development, of constructing pixel coordinates and bit-map data in source code, is not only time-consuming but inappropriate to the data being devised. Bit maps aren't numbers; they're images. Pull-down menus aren't null-terminated arrays of strings; they're pull-down menus. Text messages aren't character arrays; they're messages. By supporting the partitioning of applications into distinct types of components, Open Interface is not only speeding development, it is also improving software organization.

PORTABILITY AND COMPLIANCE A QUESTION MARK. Notwithstanding its strengths and its flexibility, a product like Open Interface sometimes goes against the grain in the Unix market. Portability is such an innate concern that programmers often shy away from third-party software libraries like those in Open Interface—after all, it doesn't run on AT&T 3B2s. (What? You don't even remember the 3B2?) Perhaps more to the point, since its implementation omits standard run-times for look-and-feel specs, it could fall out of compliance with those specs. However, we feel such concerns are really immaterial. End users are little affected by minor deviations in look-and-feel and often want the best of all worlds anyway. And scrupulous developers should think about the piles of Windows PCs and Macintoshes they can support by "compromising" their portability.

HOW FAST CAN DEVELOPERS ABSORB AND MASTER IT? Neuron Data has clearly driven toward a portable GUI model that embodies heavy emphasis on high-level abstraction. It also embodies broad and varied functionality lifted from its supported platforms. Both characteristics will create a steep learning curve for the rank-and-file developer in creating applications. Simply navigating a widget set can be a daunting experience, even in single look-and-feel GUI builders. Understanding in entirety all of Open Interface's libraries could be a very long process. The abstraction of Open Interface's add-on widgets, however, poses a different problem. The power in these widgets is veiled under an API that is deceptively simple but extremely, though subtly, rich. Neuron Data found that its Open Interface 1.0 customers didn't perceive all of the functionality they were getting, so Release 2.0 has substantial amounts of example software bundled in. Nonetheless, we feel it will take some time for developers to really leverage the capabilities of Open Interface.

IMPORTANT PIECE OF FUTURE PROGRAMMING. Open Interface goes a step beyond current programming tools. Not only does it build GUIs for multiple platforms—and very disparate ones, at that—it also structures applications to grow into new computing environments, like the workgroup-centered initiatives at Apple and Microsoft, and the distributed Unix environments like ONC and DCE. It brings GUI programming out of the text editor and lifts it above the GUI wars to where it should be: delivering functionality and usability to the end user. Open Interface 2.0 brings the promise of its first release closer to completion. GUI extensions are more thoroughly supported, the add-on widget set has been enlarged and enhanced, and niceties like PostScript printing are fleshing out the end-user capabilities available under the product. In our view, Open Interface is not only a valuable product, but it also could become the key to preserving the Unix beachhead on the desktop market. —*A. Wolfe*



Vital Information

Complete Details

When You Need It

New Format!

Special Research Reports

From Patricia Seybold Group

*Patricia Seybold's Special Research Reports give you the detail and depth of books **and** the timeliness of magazines.*

Unix Relational Database Management 1992/93

Vendor Strategies, DBMSs and Applications Development Tools

By Judith R. Davis	Full Report Available Spring, 1993	$595
	Quarterly Updates of Comparison Matrix	$595
	Report and Quarterly Comparison Matrices	$795
	One Comparison Matrix of Your Choice	$200

The third version of this well-respected and popular report continues to cover the marketing strategies of the major Unix relational database vendors: Ask/Ingres, Informix, Oracle, Sybase, Progress, and rising contender Borland/Interbase. Now for the first time, Patricia Seybold Group gives readers the ability to receive the information in this report in a more flexible and timely manner to suit a variety of research needs. Because the product comparison matrix has proved to be invaluable to readers consistently following this market, we are offering the chart as a separate *or* bundled product. Readers can still receive the chart in the full report, or they can buy the chart separately in quarterly updates.

The Full Report. This includes an in-depth analysis of corporate strategy, product strategy, and positioning for each vendor, a review of product architecture for both the database engine and the application development environment, and a product comparison matrix.

Product Comparison Matrix. The report has always included an additional section which is an exhaustive comparison matrix of the features, functions, and architectural characteristics of each product. The matrix includes such categories as file structure, database parameters, user interface capabilities, data types, indexing, forms, architecture type, SQL statements, report writers, application development tools, and more.

The Report, the Matrix, or Both. Readers can now follow the progress of the database products as the vendors make enhancements, via quarterly updates of the report's detailed matrix. The first chart update is ready this June, with one to follow each quarter. Other readers may choose to wait until the spring of 1993, to read the report in its entirety. Choose the option that aligns with your database research requirements.

- Receive the product comparison matrix section of the report separately, four times per year as we update the chart to reflect vendor product updates, OR
- Receive a matrix update each quarter and the final report in the Spring of 1993, OR
- Receive only the final report in the Spring of 1993, which includes the final matrix

For More Information, Fax (617) 742-1028 or Call (617) 742-5200.

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

IOS-792

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
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

1	Jan.	The Groupware Phenomenon—Does It Focus on the Proper Issues?
2	Feb.	Digital's TeamLinks—A Renewed Focus on the Client Desktop
3	Mar.	Requirements for Workflow—What Should We Expect from the Vendors?
4	Apr.	Desktop Multimedia—Moving beyond the Chicken and the Egg
5	May	Borland International—A Database-Centric, Object-Oriented Approach to Desktop Integration
6	June	Apple's Macintosh—Can It Become "the Cadillac of Collaboration"?
7	July	Business Intelligence—A Framework for Data Analysis Applications

UNIX in the Office

1991—Volume 6

#	Date	Title
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

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
3	Mar.	Europe's Harness Project—Integrated Technology for an Open, Object-Oriented, Distributed Applications Platform
4	Apr.	The X Window System—Where is Its Future?
5	May	HP's Master Plan—Winning Is Everything in Palo Alto
6	June	Digital's DECworld Gems—Alpha and Accessworks Shine

Open Information Systems

7	July	Integrating Applications in the Real World—Evolution, Not Revolution

Network Monitor

1991—Volume 6

#	Date	Titl
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

1	Jan.	Securing the Distributed Environment—A Question of Trust
2	Feb.	HyperDesk DOMS—A Dynamic Distributed Object Management and Applications Development System
3	Mar.	Smart Hubs—Establishing a Manageable Internet Foundation for Distributed Computing
4	Apr.	Message Express—A Message Platform for Cooperative Processing
5	May	Novell NetWare 4.0—Building toward an Enterprise Distributed Object Computing Environment
6	June	Distributed Printing—Major New Approaches Begin to Relieve One of Distributed Computing's Most Frustrating Problems

Distributed Computing Monitor

7	July	The New E-Mail APIs—Finally, Real Progress toward Mail-Enabled Applications