Distributing Data

DISTRIBUTED APPLICATIONS ARE THE NEXT STEP IN LAN CONNECTIVITY.

by Eric W. Wasiolek

There is a critical need in today's organizations to freely and transparently share any and all information, located on any and all computers in the organization. But with most organizations increasingly populated by a wider and wider variety of desktop, mini, and mainframe computers — running a variety of operating systems — the task does not look easy.

What is needed, beyond the obvious physical connections between different systems, are applications that make transparent use of those connections. These distributed applications, which can range from distributed databases and distributed file systems to networked windowing systems, are the next step in the evolution of true connectivity. And while they are still barely out of their swaddling clothes, serious distributed applications that run across a variety of computers and operating systems are beginning to appear in greater numbers. Here's an overview of the distributed application world, taking a close look at what distributed systems are available and how they work.

As the communications industry evolves, connectivity is being waged at increasingly sophisticated levels. Yesterday, just physically connecting machines of the same type to a common wire to pass information at acceptable transfer rates was an accomplishment. Now, utilities which allow the transfer of files, the ability to emulate a terminal on a remote system and the ability to share electronic mail with other systems on the network are fairly commonplace.

Today, though, the connectivity battle is being waged at the *application* level, with distributed file systems, networked windowing, distributed databases and various distributed applications products which provide transparent access between systems of dissimilar type.

What Users Want

Because of the increasing presence of many different types of computers and operating systems in today's organizations, information a user needs may well not only be located on a system *other* than theirs, but on a different *type* of system altogether. Information is increasingly dispersed across the organization on many different types of computers.

This trend is exacerbated by the strong inclination to replace terminals with desktop computers. This situation creates a serious deficiency in the organization: if users are not connected to all other systems with an ability to share information between those systems, islands of information form, preventing users from accessing all of the data they need to do their jobs.

The only remedy is to connect users to one another through a network that can connect dissimilar computers, and offer some application connectivity so that information on any one system may be accessed from any other system.

One answer to this need has been to connect groups of homogeneous users in a system like a PC LAN, and to offer gateway or terminal emulation connections to heterogeneous hosts like IBM mainframes, where most corporate information is stored. However, such a solution is clumsy at best. First, terminal emulation approaches require that the user learn the operating system and applications on the host, rather than presenting the information to the user through his familiar operating system. This inconvenience is magnified when the user needs to access information that resides on many different types of hosts. Second, gateway approaches are slow, especially for terminal emulation tasks. Plus, if data is transferred to a local host via a file transfer utility, the data is most likely in the wrong form to be useful on the local system. Either the byte ordering is wrong, or the data format is not acceptable to the local spreadsheet or database manager.

What users in today's complex heterogeneous environments require is to be able to interact with remote data and programs regardless of the type of system on which they are located, through a unified windowing interface. Ideally, the user should be able to sit at his desk and have access to information located anywhere in the organization as if the information were located on his desktop. This means that his access to that information should be quick and transparent. As an example, an Apple Macintosh user should be able to call up his favorite spreadsheet program by clicking on the appropriate icon and load a file which is actually located on an IBM mainframe, graphically displaying the spreadsheet data as if it were from a file located on his local drive. The same user should be able to request data from a VAX/VMS system, and get the information with the same level of transparency.

Though this kind of connectivity is still in its infancy, there are in fact products on the market today which offer transparent access across different systems. All of these products are comprised of distributed applications, implemented with a communications technology which connects dissimilar systems — usually a LAN. For lack of a better name, we'll call these products *heterogeneous distributed applications.*

What's Available

Heterogeneous distributed applications exist, but they are far from abundant. Part of this is due to their requirements for sophisticated software engineering and the scarcity of networks (usually based on TCP/IP or OSI) that offer the necessary point-to-point communications between dissimilar systems.

Several database vendors offer distributed implementations of their products, which allow users on one type of system to access data on another system transparently and fairly efficiently, as if the data were on the user's local hard disk. Vendors offering such systems include Gupta Technologies (Menlo Park, CA), Oracle (Belmont, CA), Informix (Menlo Park, CA), Relational Technologies (Alameda, CA) and some others.

Likewise, several vendors are creating implementations of networked windowing systems, which allow viewing the output of and interacting with programs which are actually executing on different systems located across a common network. Vendors marketing networked windowing systems include Locus Computing (Santa Monica, CA) with its XSight product line, Graphics Software Systems, with its X/Windows product, Grasshopper with its NEWS product for the Macintosh and Sun Microsystems, with its X/NEWS windowing system.

How They Work

There are two fundamental components to any heterogeneous distributed application: the distributed application and the communications technology upon which it is built. A distributed application is generally split into two or more programs, which concurrently execute on different computers and yet communicate with each other by passing messages. Typically, a distributed application is functionally split into *client* programs, which usually run on users' desktop computers, accepting their inputs and displaying the results of their requests, and *server* programs, which usually run under multiuser operating systems on larger supermicro, mini or mainframe computers, and execute user requests for resources on behalf of messages sent by the client program.

Most heterogeneous distributed applications are implemented according to this *client-server* architecture. Client programs issue request messages through a common communications interface to server programs on different types of systems, which execute the request and return information.

The distributed application, however, requires a communications subsystem to pass messages between client programs and server programs. Moreover, if client and server programs reside on different types of computers, as in our example, the distributed application requires a communications subsystem that specifically allows the passage of messages between dissimilar systems. Such a communications subsystem has three basic components: the application programming interface (API), the transport subsystem and the physical network link. The API allows the client and server programs to read and write messages to the communications subsystem. The "write" command specifies the logical name of the system to which the message is to be delivered. The "read" command listens for messages delivered from remote programs to the local program. API commands do not specify physical addresses of systems on the network; the underlying communications transport subsystem resolves the physical location of the destination system from the logical name.

It is important to note that the API isolates the client and server programs from the underlying communications transport mechanism, so that the application never needs to specify *how* a message is to be delivered to its destination, just that it *is* to be delivered to that destination. The transport subsystem accepts messages from the API, determines their physical destination and sends them to the physical network; conversely, it receives messages from the physical network and presents the message to be read by the application program through the API. Finally, the physical network converts the message into a physical signal format which may be sent over a wire to the destination system.

CLIENT-SERVER ARCHITECTURE

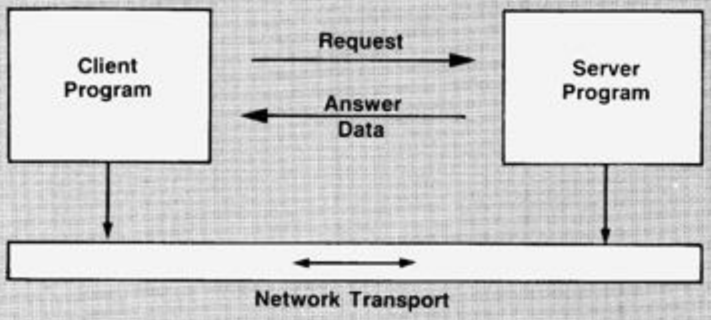

Figure 2 — *Most distributed applications today are built on the client-server model, where a client application — typically running on a desktop PC — makes a request for data over the network, and a server application — typically running on a larger host — services it.*

Distributed Databases

As an example, a Macintosh user may request information from a database which is actually located on a VAX running VMS located in a different department. In this case, the user's request is accepted by the client program, packaged into a message, most likely in Structured Query Language (SQL) format, shipped across the network to the VAX system, where a server program is in turn "listening" for requests from the client program. The server program then receives the SQL message, decodes it, executes the request which, say, is to locate all records in the database on the VAX where bicycles are of type "Schwinn," and bills are of type "unpaid." It then ships the records back to the client, which in turn displays them, perhaps in a graphic format, to the user who requested them.

Theoretically, client and server programs may exist in a many-to-many relationship, where multiple users interacting through client programs may request information located on multiple remote hosts running multiple server programs. In the case where multiple clients simultaneously request the same resources, say the same database record, the server program arbitrates access to the common resource, using the same kind of record or file locking supported by a system like DOS 3.1.

Networked Windowing Systems

There are currently two popular models of networked windowing systems: X/Windows and NEWS

(Microsoft and IBM's Presentation Manager is not a networked windowing system). Both have emerged from the UNIX world but are now being made available in heterogeneous implementations.

With networked windowing systems, the client-server model is reversed. The server program resides on the desktop system, and the client program resides on the multiprocessing host. The server program resides on the desktop because it "serves" client programs executing on various host computers across the network with a common windowing resource, namely the screen of the desktop computer. Since the window server may accept the output of many client programs simultaneously, the desktop operating system must either actually be multitasking, or at least simulate multitasking.

The two popular models for heterogeneous networked windowing systems, X/Windows and the NEWS/PostScript model, both use a reversed client-server architecture, with window servers running on desktop systems and window clients running on hosts. X/Windows is a public domain specification from M.I.T, and is the most popular networked windowing system on the market. X/Windows works by piping the output of a program running on a host computer over the network to a window on the desktop system. The program on the host is typically a graphics program.

Conversely, the desktop user may perform program control of the remote application through a window; internally, the system is sending control sequences from the window server to the window client. If the window server is simultaneously viewing the output of multiple remote window clients, the user interacts with each remote program sequentially through an "active window," which maps the user's keystrokes to the particular remote program.

Under the PostScript model, the remote client program pipes a series of PostScript commands to the local server program, which interprets them to form an image of the graphic output from the client program. In effect the remote client program sends a PostScript graphic language description program to the local server, which executes the Postscript program to form an image that is the graphical output of the remote client.

This networked windowing model has several advantages. First, it is more efficient: less network traffic is created by sending a PostScript description of the graphic image across the network than is by sending the bit-map for the graphic image, which is what X/Windows does. Second, much of the input-response loop may be handled locally, since the PostScript program is running locally. And, finally, PostScript can be sent to any PostScript device, like a laser printer or a facsimile machine.

There are a variety of X/Windows NFS client may access as an NFS file the same file, located on a VAX/VMS server, that an OS/2 LAN Manager client would access as an OS/2 file.

Some vendors are taking steps to provide a *universal* DFS — a DFS which allows a user to access files located on different types of systems using different types of DFSs — all as if the files were local. For example, with a universal DFS, a Macintosh user running on an AppleTalk LAN should be able to access a file located on a UNIX system running LM/X, as if the file were a local Mac file. Internally, such a universal DFS must resolve DFS protocol differences between AppleTalk and LM/X, as well as send the disk access request and returned file between hosts running dissimilar operating systems.

TWO MODELS OF NETWORKED WINDOWING SYSTEMS

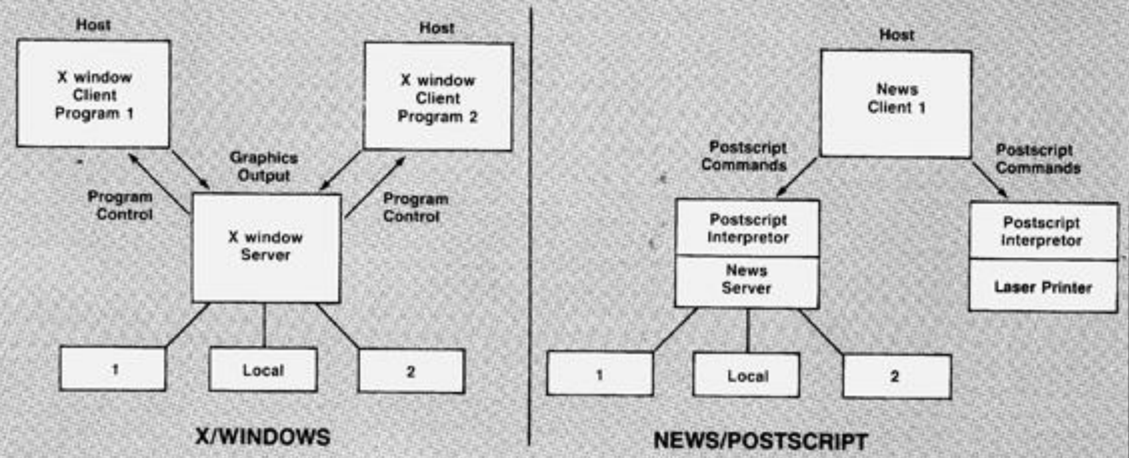

Figure 3 — *The two popular models for heterogeneous networked windowing systems, X/Windows and the NEWS/PostScript model, both use a reversed client-server architecture, with window servers running on desktop systems and window clients running on hosts. X/Windows works by piping the output of a program running on a host computer over the network to a window on the desktop system. Under the PostScript model, the remote client program pipes a series of PostScript commands to the local server program, which interprets them to form an image of the graphic output from the client program. The PostScript model tends to be more efficient than the X/Windows approach.*

Other Distributed Applications

The applications discussed so far are the most common examples of heterogeneous distributed applications. But they are by no means the only types. Actually, the standard suite of TCP/IP and OSI applications are themselves heterogeneous distributed applications, including SMTP or X.100 for networked mail, TELNET or VTP for terminal emulation, FTP or FTAM for file transfer and others. Each of these standard applications includes a client and a server portion which may reside on dissimilar computers. Other custom applications are emerging, such as Applix's ALIS integrated office automation package, which allows accessing remote spreadsheet, textual, graphic or database data from within a multiwindowing environment on the local desktop. Heterogeneous distributed applications also have the capability to unite diverse hardware. For example, a server program could collect hardware signals from shop floor instruments, storing them in a local database and offering remote clients access to the information.

How It All Relates

Most heterogeneous distributed applications are built according to the client-server model. The client and server portions of the application send messages and data between each other through a common communications link across heterogeneous hosts.

It is possible for one type of distributed technology to use another. For example, a heterogeneous distributed database program may be implemented by having non-distributed database programs make file system calls to a heterogeneous distributed file system. In such an implementation, the database simply makes local file system calls, and if the files it needs to access are located remotely on a dissimilar system, the heterogeneous DFS locates and NEWS/Postscript products on the market. The X.11 protocol standard of X/Windows, is the industry standard for UNIX systems and it is migrating to other platforms. Locus Computing has implemented an X/Window server for DOS with a product called PC-XSight. Locus also markets a UNIX V.3 '386-based X/Window server which allows the user to run a local DOS task under Merge-386 (virtual DOS under UNIX) in one window, while interacting with other remote programs in other windows. Sun Microsystems has a networked windowing system called NEWS, which is based on the Postscript model. However, Sun will soon release an "X/NEWS" server which allows concurrent interaction between X/Windows- and Postscript-based programs across the network.

A "UNIVERSAL" DISTRIBUTED FILE SYSTEM

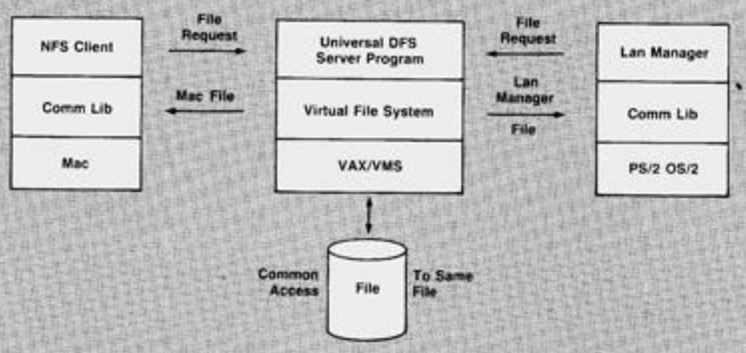

Figure 4 — *In a "universal" distributed file system (DFS), different computers running different operating systems can transparently share the same files. Here, a Macintosh and an OS/2-based PS/2 running LAN Manager both can access the same file; to the Mac it looks like one of its own local files, while to the PS/2, it looks like a standard OS/2 file.*

Distributed File Systems

A *heterogeneous distributed file system* is a distributed application which allows users to access remote files located on different types of systems, as if they were local files. There are three such file systems available now: Sun Microsystems' NFS, available across MS-DOS (licensed as PC-NFS from Sun and Locus Computing), VMS (from Excelan and Wollongong), Macintosh (from the University of Michigan) and many varieties of UNIX (through Lachman Associates); MS-Net (from Microsoft), otherwise known as the Server Message Block (SMB) protocols, available across PC- and MS-DOS, VMS (from Syntax Corporation), Xenix (marketed as Xenix-NET from the Santa Cruz Operation) and various versions of UNIX; and the forthcoming FTAM from the OSI protocols, specified to run across all systems that implement it. Additionally, TOPS A Sun Microsystems Company's TOPS, Apple's AppleShare and Microsoft's OS/2 LAN Manager (and the LM/X version for UNIX) are available in limited non-native environments.

Each distributed file system (DFS) consists of a DFS client and a DFS server. Say a user on a DOS system running PC-NFS — a DFS client — wants a file that resides on a VAX/VMS system, running VMS/NFS — a DFS server. The PC-NFS client redirects the DOS file system call across the network to be executed by the VMS/NFS server. The server locates the file and sends it across the network. If all of this is done efficiently, i.e., with caching and across a fast network medium like Ethernet, the remote file access is indistinguishable to the user from a local file access.

In a distributed file system, client programs from different DFS protocol families may transparently share files on a remote host. For example, a Macintosh the file and returns it to the local database application. Likewise, vertical or database software, distributed or not, may send its screen output to the window server on the desktop. This would allow a user to simultaneously view database data in one window concurrent with other local and remote task output in other windows, on the same screen.

The Future

The emerging technologies discussed in this article provide new vistas of development. Much work yet remains in truly integrating heterogeneous distributed technologies to present users with a truly unified and transparent window to resources located on different systems across the network. ❑

EXCELAN

Eric Wasiolek is product marketing manager for UNIX products at departmental LAN vendor Excelan (San Jose, CA).