2

WSRC-IM--90-83-1

DE92 009964

Reviews of Computing Technology:

Client-Server Technology (U)

WSRC-IM-90-83-1

September 1, 1990

Westinghouse Savannah River Company
P. O. Box 616
Aiken, SC 29802

SAVANNAH RIVER SITE

Reviews of Computing Technology:

Client-Server Technology (U)

by S. M. Johnson

WSRC-IM-90-83-1

September 1, 1990

Authorized Derivative Classifier

G. F. Hayhoe

Westinghouse Savannah River Company
P. O. Box 616
Aiken, SC 29802

Table of Contents

1 Introduction

One of the most frequently heard terms in the computer industry these days is "client-server." There is much misinformation available on the topic, and competitive pressures on software vendors have led to a great deal of hype with little in the way of supporting products. The purpose of this document is to explain what is meant by client-server applications, why the Advanced Technology and Architecture (ATA) section of the Information Resource Management (IRM) Department sees this emerging technology as key for computer applications during the next ten years, and what ATA sees as the existing standards and products available today. Because of the relative immaturity of existing client-server products, IRM is not yet guidelining any specific client-server products, except those that are components of guidelined data communications products (see WSRC-IM-90-81-1) or database management systems (see WSRC-IM-90-81-2).

Client-Server Defined

The basic idea behind client-server technology is that, because users now have intelligent workstations on their desktops, computer applications or information systems can be partitioned into two components. The client component is software that runs on the desktop computer and handles interactions between the application and the person using the system (the user interface). This client software may also contain part of the application's logic. However, the client software must use a service component to complete the application. This service component, or server, is software which runs on another computer system and which communicates to the client via a computer network. The notion of a server is a logical one; there may be several service tasks running on a general purpose machine. Often, the server computer is dedicated to its service function. For example, a server may be a database management system which accepts requests for data records from multiple clients and returns the requested data to the clients or updates the database with client-supplied data.

In general, client-server applications work in the following way (see Figure 1). The user starts the client program on his or her workstation. This soft-

ware interacts with the user, navigating through the program options and getting information from the user regarding what is to be done. When the client software needs information or processing steps which it cannot independently satisfy, it uses the computer network to connect to a server program on some other computer system. The client then requests the data or function that the server satisfies. The person using the application is usually unaware of the client-server interaction. Data which the client obtains from the server is presented to the human user as if only the workstation were involved.

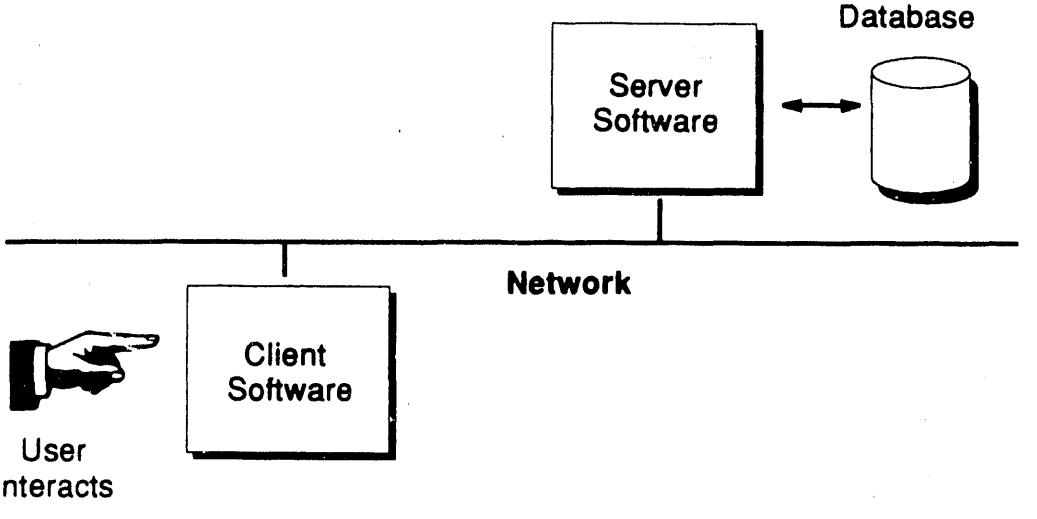

Figure 1: A typical client-server application

Why would software vendors (and users) want to use this approach? After all, it means that developers must produce more complicated applications for the computer network, and at least two computer systems are involved in each application. There are two basic reasons that the client-server approach is attractive. First, because the user, the human operator, deals only with the client software directly, the user can be given a powerful interface into the application that is consistent with his or her view of the workstation. Thus, the user must learn only one way of "dealing with the computer."

A simple example of our current situation can make this benefit clearer. Today, a user of the Apple Macintosh (or OS/2 Presentation Manager, NeXT, or Sun OS) sees a consistent, state-of-the-art graphical user interface built around windows on the screen and makes selections via a pointer or mouse. If that same user wants to use electronic mail via ALL-IN-1 or check purchase orders via the Procurement Cycle System (PCS), then he or she must use a host computer terminal emulation product on the local workstation, use the

network to connect to the appropriate host system, log in, and deal with the host system on its terms. That is, the user must be able to navigate through the menu options in ALL-IN-1 or PCS and know how to request functions. Thus, the user must learn three different interfaces, the workstation's native interface along with the different interfaces of ALL-IN-1 and PCS.

How much easier for the user if the interface for all applications were the same. In our example above, the Macintosh user would have a Macintosh program which would serve as a client into electronic mail. This client software would have the same look and feel of any other Macintosh program but would communicate mail messages with a host "mail server," which in turn would deliver the mail to other "clients." Our user would get purchasing data much the same way. From the Macintosh client interface, he or she would fill out a form requesting purchase order data. The client software on the Macintosh would communicate this request to a server process on the IBM mainframe (where PCS data is kept) and return the requested data. The client software on the Macintosh would display the data on the user's screen. The user would learn only one interface, the one native to the workstation. The interface would become so familiar that necessary actions would often seem intuitive.

There is a second reason that client-server applications are becoming attractive. Since the user has a powerful desktop computer dedicated to his or her use, the client portion of the application can utilize this power, and only the server portion of the application needs to use a resource shared with others. Because of this dedicated desktop power, the user interface can be more powerful and more responsive than an interface tied to a remote host system. Every application includes some functions that can be accomplished independently on the desktop. For example, if there were an application in which data stored in the database were a function of user input data, then the calculations to obtain this derived data could be accomplished on the workstation and thus keep some work off the shared service computer.

Types of Client-Server Applications

There are many variations on client-server computing. Generally, though, client-server functions can be categorized into four types:

- database requests
- communication pipelines
- remote procedures
- graphical user interface

Each of these is explained below. In the sections that follow, the various commercial products which implement these types of client-servers are also described.

2 Database Requests

A database server manages a computerized database and responds to requests from clients to update the database or return data from the database to the client. Sometimes, the database server runs on a dedicated machine, indeed may use specialized hardware optimized for database access. Often, however, the server is a process which runs on a general purpose computer system. The server must be designed to respond to requests from multiple clients simultaneously. Client computers can be varied; all that is required is that the client computer properly encode a request for data and be able to communicate this request across the network to the server. In this case, we say that the client computer shares common communications and data access protocols with the server. Thus, for example, the database server could be a process (and associated database management system) on an IBM mainframe or departmental VAX/VMS system, while the clients could be OS/2, UNIX, Macintosh, or MS-DOS systems. The protocol or language for building client-server database applications is called SQL (Structured Query Language).

SQL is an English-like query language that provides communications syntax between the "front-end" application on the client and the "back-end" database on the server. SQL is a vital application layer protocol for extending client-server applications beyond a single vendor's platform.

Both the American National Standards Institute (ANSI) and the International Organization for Standardization (ISO) are involved in developing SQL as a standard for database connectivity between various hardware platforms. SQL was developed by IBM at its San Jose, CA, research center in 1976. While all SQL products conform to the ANSI standard, the addition of vendor-specific features, or so-called extensions, could mitigate the promise of portability and interoperability. However, a group of SQL vendors have formed the Open Access Group, which they hope will be a forum for agreement on SQL implementation.

Many vendors sell database products advertised as SQL-compliant. All of these products are designed around the same SQL standard, but to add flexibility and enhance performance, each vendor has added extensions to its products. A customer choosing to take advantage of the extensions will lose portability across vendor platforms. A user deciding not to use the extensions will sacrifice performance and flexibility. Unless portability is worth giving

up performance and flexibility, customers will certainly want to use the extensions provided in the various vendors' implementations. Standard or barebones SQL provides commands for basic query, data manipulation, and configuration of relational databases. The extensions are commands that let the user take advantage of such special features of the server as advanced security or triggers. Some of the standard commands might have additional parameters beyond what is found in standard SQL. An example would be adding index creation information to a **CREATE TABLE** command, so that both the table and the index could be created in one command. With standard SQL, this procedure would require two commands: a **CREATE TABLE** instruction followed by a **CREATE INDEX** instruction.

These extensions all take advantage of features that improve the server and make it different from and supposedly more powerful than its competitors. Ignoring the extensions often means giving up the features that make a particular product strong. The user who does so is left with a mediocre product, no matter which one is chosen.

Most programmers will not write SQL statements directly but will use tools which automatically generate SQL statements. These tools will take the form of Fourth Generation Languages (4GLs) or Computer Aided Software Engineering (CASE) software.

The current generation of client-server SQL database products allows a workstation and a database server to share in the data manipulation chores in varying degrees. Another phase of the client-server database evolution consists of distributed databases that allow a single front-end program to query a back-end database, which in turn queries other engines on other machines. In such databases, data is stored in multiple servers on different types of computers, and users have transparent access from their local applications. Several manufacturers have released papers describing how this process would work, but few actual products exist.

Product Information

There are a number of SQL database products on the market for machines ranging from microcomputer-based servers to large mainframe systems. Interoperability of these products is dependent on the extensions they make to SQL and the network transport protocols they support. Micro-based SQL database servers commonly support IBM's NETBIOS protocol or Novell's IPX protocol. These protocols were developed for IBM PCs and are only recently being supported on larger machines. Minicomputer servers are more likely to support TCP/IP than one of the PC protocols. TCP/IP is becoming the common link between different server products from micros to mainframes.

The following table lists some of the SQL databases with the most relevance to the SRS computing environment and the operating systems and protocols they support:

SQL Database Products			
	Protocols Supported	**Operating Systems Supported**	**Other**
DB2	SNA TCP/IP* (June 1990)	IBM MVS	Official SAA database component
Oracle	TCP/IP DECnet, IPX/SPX, Named Pipes	IBM MVS, UNIX, VAX VMS, DOS, OS/2, Mac OS	Supports wide range of systems
OS/2 EE 1.2 Database Manager	APPC NETBIOS	OS/2 Communications Manager	Official SAA database component
Netware SQL	IPX/SPX TCP/IP* OSI*	Advanced Netware 2.1X	Industry leader for PC LANs
*Announced but not yet available			

This technology is relatively new. Vendors are striving to develop products with features that differentiate them from their competitors. Interoperability of these products with each other is not assured, and indeed, today, one can almost count on these products not interoperating with each other. Standards development and product shake-out will improve this situation in the future, but it will be a slow process. ATA will closely track developments in this area.

3 Communication Pipelines

Remote procedure calls depend on the establishment of a fixed set of procedures that can be executed on a remote system by the clients. Often, an application will require the cooperation of two computer systems where the interaction is viewed as a conversation in which data is passed between the two systems. This process is commonly referred to as task-to-task communications. Communications protocols allow the establishment of two unidirectional data "pipelines." Each system uses one of these pipelines to write messages to the other system and reads messages from the other system from its pipeline. In this way, a programmer can build an application by writing cooperative codes that execute on two different systems at the same time.

IBM APPC

Advanced Program-to-Program Communication (APPC) is an IBM architecture which enables programs on networked computers to exchange information. APPC is designed for program-to-program, peer-to-peer communication between both similar and dissimilar IBM machines. APPC is part of IBM's Systems Network Architecture (SNA) as a particular type of Logical Unit (LU), LU6.2, supported by Physical Unit PU2.1. It is the key component of IBM's Systems Applications Architecture (SAA). APPC's pipelines are the mechanism for achieving cooperative processing in the context of SAA.

LUs communicate over logical connections called sessions. When a program wishes to communicate with another program, it asks the LU for temporary, exclusive use of a session. Many programs may take turns using a single session; thus, they are said to share a session serially, or to time-slice the session. Each time-slice is called a conversation. The advantage of this approach is that many conversations may be allocated and deallocated without establishing and tearing down multiple sessions, a time-consuming procedure. A session is typically a long-term connection, while a conversation is usually much shorter.

Conversations may be basic or mapped. Basic conversations allow transaction programs to transfer data with only a 2 byte length prefix and are typically

used by SNA-defined service programs. They are more flexible than mapped conversations but are more difficult to use. Mapped conversations put data into a standard format for transmission and allow transformations called maps to be performed on the data. They are higher level than basic conversations and are generally used for user program-to-program communication.

DECnet Objects

A DECnet object is a DECnet application layer process with which the user can connect over a logical link to perform general-purpose network services. DECnet objects are accessed by name or number at the time the connection is made. Once a connection is made to an object, a reliable bidirectional data stream can be opened between the two machines. This data stream can maintain record boundaries or be an unformatted bit stream, depending on user requirements.

DECnet provides the capability of sending out-of-band data between the two connected processes. Out-of-band data is data outside the normal data path and is typically used for control functions. DECnet task-to-task communications are supported on IBM PCs, IBM mainframes, Sun and DEC UNIX, as well as DEC VAX systems.

The ALL-IN-1 system at SRS contains a client-server application, VideoText (VTX), which features DECnet objects (see Figure 2).

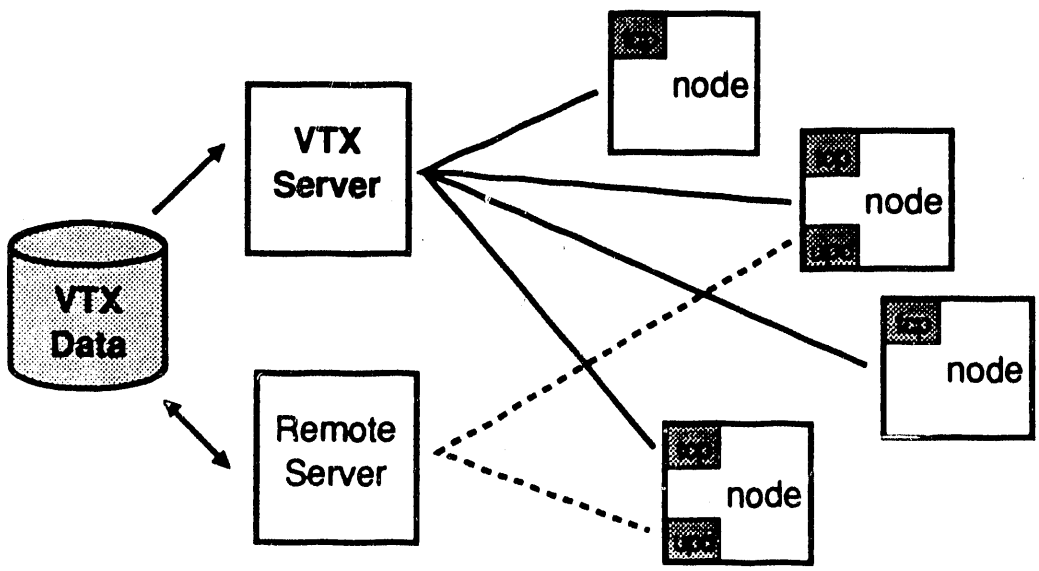

Figure 2: SRS VideoText system

VideoText is a type of bulletin board service that gives users access to frequently changing data of general interest, such as daily weather reports. VideoText maintains a single database of ASCII "pages" on a VAX/VMS system which may be accessed by VideoText clients anywhere on the network. On the same system with the database is a server process which reads the database file and delivers VideoText pages to VideoText clients on request. VideoText clients, called Terminal Control Programs (TCPs), make requests for VideoText pages via DECnet objects. The pages requested are sent to the TCP which displays them on users' screens. Another process serves as an update server. The VideoText page database is remotely updated by clients which communicate with the VideoText update server. The update server writes new and changed pages into the database.

The Berkeley (UNIX) Socket Abstraction

Both Sun RPC and NCS use extended versions of the Berkeley UNIX socket abstraction for application-to-application communication.

Version 4.2 of the Berkeley UNIX kernel introduced the concept of sockets as an interprocess communications mechanism. A socket is an endpoint of communication referred to by a descriptor. A descriptor is treated programmatically just like a file. Two processes can each create a socket, and then connect those two endpoints to produce a reliable byte stream. Once connected, the descriptors for the sockets can be read or written by processes. The transparency of sockets allows connection of the output of one process to the input of another process. In the case of remote procedure calls, these processes may reside on different machines. A socket system call creates the socket and returns a descriptor. Each socket has a type that defines its communications semantics; these include properties such as reliability, ordering, and prevention of message duplication.

Each socket has a communications protocol associated with it. This protocol provides the semantics required by the socket type. Applications may request a specific protocol when creating a socket or may allow the system to select a protocol that is appropriate for the type of socket being created.

While sockets were developed specifically for the Berkeley versions of UNIX, the concept has been used in other, unrelated environments. Apple has implemented sockets as an interface to their AppleTalk network protocol suite.

APPC, DECnet objects, and sockets are the most significant communications pipelines available today. DECnet objects and APPC are important because they are the cornerstones of their respective corporate architecture initiatives. Sockets are important because they are generally available for use with TCP/IP implementations, and TCP/IP is the de facto standard for multi-vendor network communications protocols. APPC and DECnet will continue to be important communications pipeline architectures for some time because of

their large bases of installed systems. They will both be available on several platforms other than their native ones, but neither will ever be truly "open." Sockets are a much more open solution but will eventually be supplanted by an OSI (Open Systems Interconnect—the open network protocol suite soon to be required by the Federal Government) mechanism, as will APPC and DECnet objects.

4 Remote Procedures

Client-server interactions can be established to let client software call for the execution of procedures; you can think of these as remote subroutine calls which run on a server system. The client must follow the appropriate calling syntax required by the procedure, use network facilities to initiate the remote procedure, and receive the results of the procedure. For example, a supercomputer has a procedure for inverting large matrices. Since the calculation time on a personal workstation could be very long, the workstation software uses standardized remote procedure-calling protocols, passes the input matrix to the supercomputer server, and receives the inverted matrix back from the server. As before, no restriction requires the client and server computers to be the same type of system. One server can service many different clients if they all share common communications and access protocols.

One complication in client-server interactions must be considered if clients have a different computer architecture than servers. Data representations (especially floating point numbers) vary among the various computer manufacturers. Thus, if a client passes data to a server or receives data from a server, the client-server access protocol must specify data representation requirements or conventions.

Remote Procedure Call (RPC) from Sun Microsystems (upon which Sun's popular Network File System [NFS] is built) and Apollo/Hewlett-Packard's Network Computing System (NCS) are the only widely supported RPC mechanisms available today. Both have been licensed and implemented on a wide variety of platforms. Both of these products originated in UNIX environments; therefore, their greatest use is in UNIX shops. However, both protocols have been licensed by both DEC and IBM for use in their proprietary operating system environments. Sun and Apollo would each like their approach to be the de facto standard for remote procedure calls. At this time, it is not clear which method will become the standard; NCS has a richer set of features, but Sun RPC is more widely used. The choices made by the Open Software Foundation and the IEEE POSIX (1003) committee will help determine the eventual standard.

The Sun RPC Model

The Sun RPC protocol utilizes the Berkeley socket model previously described. Although Sun RPC is based on the socket model, it does not require the programmer to be familiar with the details of socket operations. High-level library functions are available for establishing and maintaining connections and for performing many other common functions.

It is necessary to define a number of terms to describe RPC. A server is a machine on which some number of network services are implemented. The machine need not be dedicated to providing remote services. It may be a general purpose computer which also provides remote services, or it may be a workstation. Services are collections of one or more remote programs. A remote program implements one or more remote procedures. Network clients are pieces of software that initiate remote procedure calls to services. Clients and servers do not necessarily reside on different machines; the same machine may implement client and server processes for different applications.

The remote procedure call model is similar to the local procedure call model used every day by COBOL and FORTRAN programmers. In the local case, the caller places arguments to a procedure in some well-specified place. It then transfers control to the procedure and eventually gains back control. At that point, the results of the procedure are extracted from the well-specified location, and the caller resumes execution.

The remote call is similar, except that one thread of control winds through two processes—the caller's and the server's. The caller sends a call message to the server process and waits for a reply message. The call message contains the procedure parameters and other information. The reply message contains the results. Once the reply message is received, the results are extracted, and the caller's execution is resumed. This flow of control is illustrated in Figure 3 on the following page.

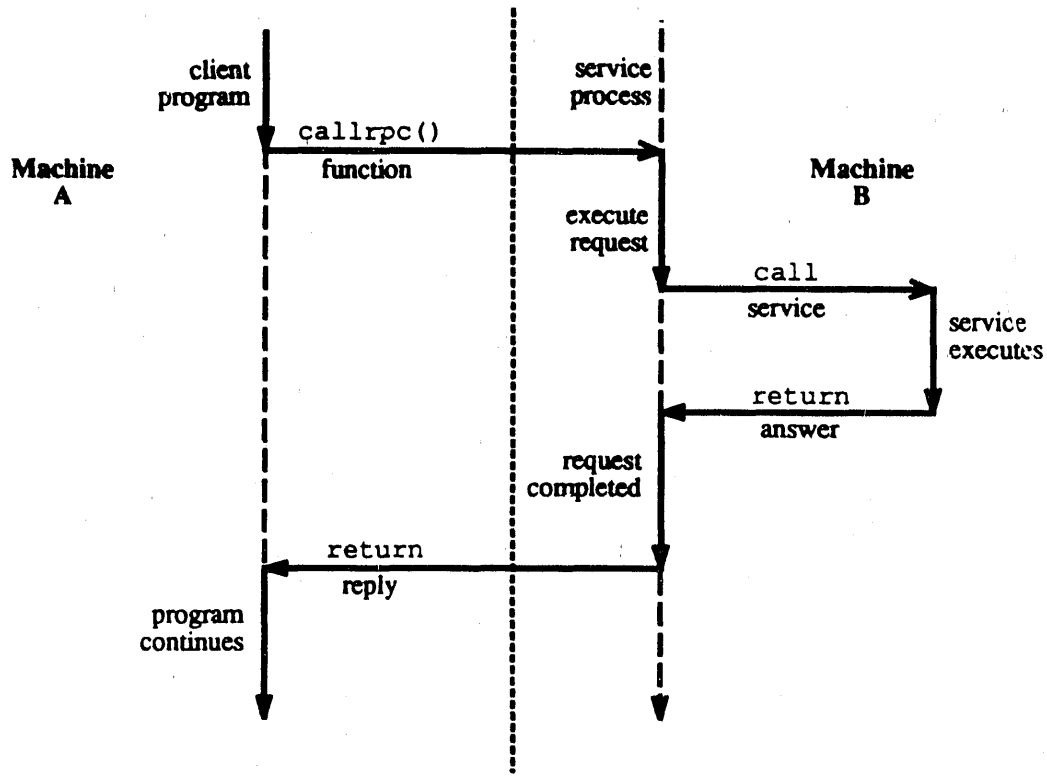

Figure 3: RPC flow of control

On the server side, a process is dormant awaiting the arrival of a call message. When one arrives, the server process extracts the procedure's parameters, computes the results, sends a reply message, and then awaits the next call message. In this model, only one process is active at any given time. The RPC protocol does not explicitly support multi-threading of caller or server processes.

The RPC architecture is designed so that clients send a call message and wait for servers to reply that the call succeeded. This design implies that clients do not compute while servers are processing a call. Such an approach is inefficient if the client does not want or need an acknowledgment for every message sent. A batching mechanism allows buffering of RPC calls when no data is returned.

Sun RPC uses the User Datagram Protocol (UDP) of the TCP/IP protocol suite by default. UDP does not guarantee reliable, in-order delivery of data. If guaranteed and reliable data transport is required, TCP can be used as the transport protocol. The TCP protocol ensures that the RPC data is transfered

in order and without error. Use of TCP for transport does require that the user program at the socket level. RPC is only implemented on top of the TCP/IP protocols, even though RPC could be implemented using datagram and reliable duplex byte stream features of other protocol suites.

On Sun systems, RPC programs are written using the RPCGEN utility, which is a compiler implementing the RPC language, which is similar to C. The RPCGEN compiler produces C source code which includes stub versions of client routines, a server skeleton, and external data representation (XDR) filters. The client stubs interface with the RPC library and effectively hide the network from their callers. The server stub similarly hides the network from the server procedures that are to be linked with remote clients. The developer writes server procedures—in any language—and links them with the server skeleton. To use a remote program, a programmer writes an ordinary main program that makes local procedure calls to the client stubs produced by RPCGEN. Linking this program to the RPCGEN-generated stubs produces an executable program.

RPC is one of the building blocks of Sun's Network File System (NFS) which has been ported to many different environments. RPC and NFS are currently available on many flavors of UNIX, VAX VMS, and IBM MVS operating systems. It is also available for the PC-DOS environment (PC-NFS). Neither RPC or NFS is currently available for the Macintosh operating system. However, it is available for A/UX, Apple's version of UNIX.

SRS RPC Development

The Scientific Computations Section in SRL has developed an application which uses RPC as both a control executive and data transfer agent for a reactor simulation model. The application, Nuclear Plant Analyzer, uses a simulation code (TRAC) developed at Los Alamos National Laboratory. This code has been modified by adding less than 100 lines of code which allows it to be controlled by an executive running on another machine. The control executive is a SunView or XView (X Windows) client running on a Sun or Macintosh UNIX workstation, while the simulator runs on the Cray XMP. Results of the simulation are translated and sent back to the controlling workstation by processes cooperating via RPC. The control code for the Nuclear Plant Analyzer is written in a general style that could be easily adapted to other simulator codes. This is an excellent example of how an application can be "distributed" with only a small amount of code development.

The Apollo (Hewlett-Packard) Network Computing System

Apollo's Network Computing System (NCS) product includes three major components: a remote procedure call (RPC) facility designed for portability and network independence, a compiler that converts high-level interface descriptions of remote procedures into portable C-language source code, and a set of software tools that let applications determine at runtime which computers can provide the services they require.

NCS is an open system written in C (source licenses are available from Apollo) and based on standard protocols. To ensure network independence, NCS uses the low-level datagram services available in most networking protocols. Apollo implements NCS over UDP and their proprietary Domain DDS protocol.

The NCS RPC uses the Berkeley socket abstraction for interprocess communication. Apollo extends the socket model with a user-mode subroutine library to compensate for different network protocols and operating systems. The RPC portion of NCS is very similar to that of Sun. It offers the same functional ability to execute subroutines on other machines; the chief differences are in implementation philosophy.

The NCS RPC runtime environment does its own error handling to ensure independence of network protocols. This environment also handles byte order conversion and differences in floating point representation. The Apollo philosophy, in contrast to Sun's RPC, is that it is better to handle these issues than to rely on the possibly different mechanisms provided by higher-level network protocols. Apollo's motivation for taking this approach was to make it easier to implement NCS with different transport protocols (TCP/IP and Domain DDS).

NCS programs are written using the Network Interface Description Language (NIDL) which supports C and Pascal syntax. The NIDL compiler produces C source code, which is then compiled on the target machine. The NIDL compiler is a pre-compiler which allows the programmer to operate at a higher level for dealing with remote procedures. The NIDL compiler is very similar to Sun's RPCGEN utility.

The NIDL compiler supports three types of binding between the program and its remote procedures: explicit binding, implicit binding, and automatic binding. In explicit binding, the NIDL specification states exactly which host to use, and this host is always used when the application is run. In implicit binding, the client establishes the binding as a variable before making any remote procedure calls. The application can query the location broker (explained below) and establish the binding between local and remote routines. The local

machine does not need to know which remote machine will execute the procedure or even where the location broker is. Explicit and implicit binding are also supported by the Sun RPC protocol.

Procedures that need to access different hosts at different times are handled through automatic binding. Each time the procedure is invoked, the local routine makes a call to find the network address of the object to be accessed and then makes the proper binding.

To eliminate the need for any network information within a routine, NCS can match available services to clients' needs through the location broker. The location broker listens on the network for services to register their capabilities. At runtime, client routines query the broker to determine which hosts to use for particular RPC calls. This is an object-oriented approach, since RPC calls are treated as operations on objects, not as calls to particular machines or server processes. No network information needs to be included in the source code.

Hewlett-Packard, IBM, DEC, and others have jointly proposed a version of NCS called DECORUM to the Open Systems Foundation (OSF) as the standard RPC mechanism for their operating system, OSF/1. Because OSF is widely supported, the choices it makes will essentially be de facto standards.

No work has been done with NCS at SRS to date.

5 Graphical User Interface

One particular style of client-server application allows application developers to build applications where the program logic, data access capabilities, and user interface are under the control of a host computer system, but the user interface may be presented on any of several kinds of desktop computer. In this case, the relationship of client and server computer is reversed from the normal viewpoint. The desktop computer provides a "display server" for the host-based application client. The major advantage of this approach is that the control of the graphics display (the screen) and the interaction of the user with input devices (keyboard, mouse, etc.) is dependent on the actual hardware. If each desktop has a graphics server capability which will take generic commands across the network from other systems, then the host application can be used with a wide variety of devices without requiring display device dependence in the application code. It is important to note that this is fundamentally different from the way in which most proprietary graphical user interfaces, such as Presentation Manager, are implemented. Presentation Manager and the Macintosh operating system tightly couple the program handling the user interface to the desktop machine.

X Windows

The predominant example of this technology is the X Window System, or X, a network-transparent window system developed at MIT in 1984. Since then, several versions have been developed, the most recent of which is X Version 11.4 (X11). X11 has been adopted as an industry-standard windowing system. X is supported by a consortium of industry leaders such as DEC, Hewlett-Packard, Sun, IBM, and AT&T that have united to direct, contribute to, and fund the continuing development of X. Almost all workstation vendors have accepted the X Window System as a standard interface for their workstation hardware. In addition to the system software development directed by the consortium, many independent developers are producing application software for use with X.

The X Window System architecture is divided into two distinct parts (see Figure 4).

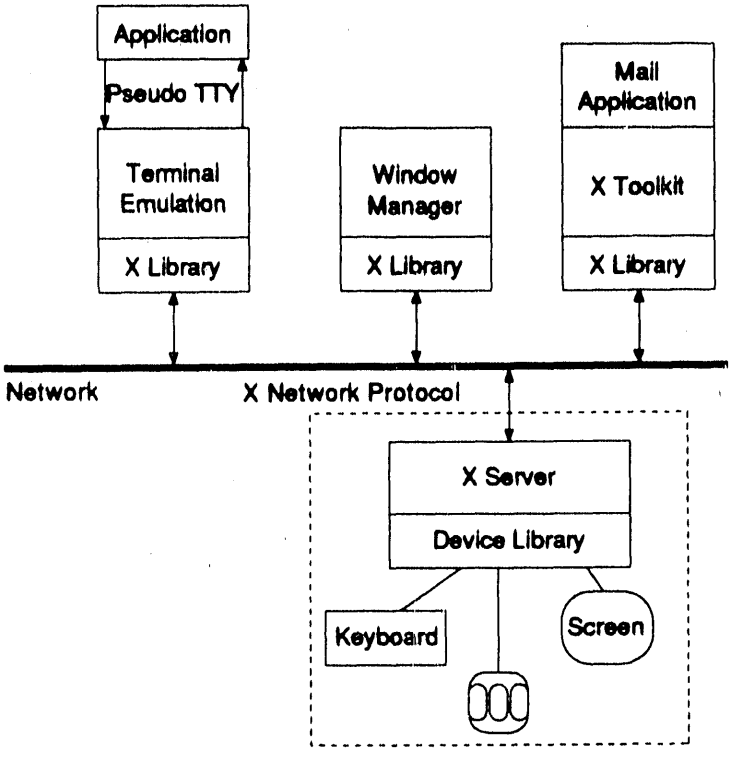

Figure 4: The X Window System architecture

Display servers provide display capabilities and keep track of user input. Clients are application programs that perform specific tasks. This division within the X architecture allows the clients and display server either to work together on the same system or to be separated over a network. For example, a relatively low-powered PC or workstation (an Apple Macintosh or IBM PS/2) might be used as a display server to interact with clients running on a more powerful system (the Cray XMP/EA). Even though the client program is actually running on the more powerful system, all user input and displayed output occur on the PC or workstation server and are communicated across the network using the X protocol.

It is important to remember that with X, the notions of client and server are the reverse of those used in most client-server configurations. The server is at the desktop, "serving" the display to clients who may reside on the same machine or across the network.

The X display server is a program that keeps track of all input coming from input devices, such as the keyboard and mouse, and input from any clients that are running. As the display server receives information from a client, it updates the appropriate window on the display. The display server may run on the same computer as a client or on an entirely different machine.

X allows a user to run many clients simultaneously. For example, the user can run a database producing a pie-chart, display a menu of options, and run

a text-editing session. While these programs may display their results and take input from a single display server, they may each run on a different computer on the network. It is important to note that the same programs may not look and act the same on different servers since there is no standard user interface, because users can customize X clients differently on each server, and because the display hardware on each server may be different.

A window manager is a client that allows the user to specify the sizes and positions of windows on the display. With window managers, the user can move and resize windows, rearrange the order of windows in the window stack, create additional windows, and convert windows into icons. Individual vendors are creating their own window managers—for example, DECWindows from DEC features Session Manager.

X as a Standard

X Windows is often touted as the standard for user interfaces. That statement is true as far as it goes. The X network protocol has become a de facto industry standard for Graphical User Interfaces (GUI) at the workstation and departmental computing level. At the PC level (IBM, Apple, NeXT), however, proprietary interfaces dominate. The standards issue for workstation interfaces is a murky one because X Windows alone does not completely define the user interface. In fact, X is purposely designed not to constrain the actual look and feel of the interface. A GUI standard built on top of X Windows requires two other parts: the programming toolkit used to develop client applications and the window manager. At least two competing organizations are trying to promote their products as de facto GUI standards based on X Windows: the Open Software Foundation Motif interface (OSF/Motif) and Open Look . Both of these products are based on X Windows and will interoperate to a certain degree.

At present, it appears that OSF/Motif will be the dominant X-based GUI. The major workstation vendors (except for SUN) are members of OSF and support OSF/Motif. DEC, IBM and HP/Apollo are expected to offer OSF/Motif as the standard GUI for their workstations within the next two years. IBM recently announced the RISC System/6000 series machines with OSF/Motif as a standard interface. Motif implements the "look and feel" of IBM's Presentation Manager (PM) window GUI. PM is a key component of IBM's Systems Application Architecture.

X Windows server-only implementations are available for the DOS and Macintosh environments. The DOS implementation allows the DOS machine to act as an X server only, while the Macintosh version allows concurrent operation of the Macintosh operating system and X Windows. X client applications can share screen space with Macintosh windows and have the look and feel of Macintosh operating system windows, or they can maintain a separate "virtual" screen with a different look-and-feel (for example, Motif). Because DOS

and the Macintosh operating system are not true multitasking operating systems, client applications must run on different machines. It is important to note that X demands a large amount of memory which it allocates dynamically. Problems may arise when physical memory limits are reached because DOS and the Macintosh operating system do not support virtual memory. Memory limit and allocation problems with X are not fundamental limitations, however. X version 11, Release 4 (released January 1990) will address some of the memory allocation issues for all machines. Meanwhile, Macintosh System 7.0 will soon provide virtual memory for the Macintosh, and OS/2-based X servers will soon be available.

A new class of terminal input device, the X Windows terminal, has also been created. An X terminal is a device that supports a bit-mapped display, keyboard, and mouse. It has no disk and has only a rudimentry operating system typically contained in ROM. It allows enough user interaction to make a connection to a remote computer and start an X client application. Once an initial connection has been made, most user interaction will occur directly with client applications located on computers across the network. As more and more applications require bit-mapped display devices, X terminals will probably take the place of the VT100 alphanumeric terminal as the universal user display device. Both DEC and IBM as well as several smaller vendors offer an X terminal. These devices require a LAN interface (Ethernet or Token Ring) and cost more than an alphanumeric terminal but about the same as a low-end personal computer. X terminals will be subject to the same memory limits as current personal computers but will probably perform X functions better than personal computers in the same price range running X Windows server software.

6 Conclusions

The SRS computing environment consists of a variety of platforms running both vendor and locally developed codes. Data often resides in different places on different hardware architectures and operating systems. It is desirable to provide better quality assurance for this data and better utilize our diverse computing resources. ATA expects that client-server applications will accomplish this goal by providing a relatively uniform user interface across hardware platforms and facilitating efficient sharing of computing resources over LANs. To do this will require intermachine communications of all of the four types discussed here.

It is inevitable that client-server applications will proliferate as the trend towards multi-machine computing environments continues. It is also clear that large sites like SRS will have machines from two or more vendors which they will want to interoperate smoothly and efficiently. The first step towards this interoperation is the selection of appropriate standards for support of the client-server model.

At this point, no dominant standards have yet emerged in the four areas discussed. Various standards committees and the marketplace itself are working to determine true standards in these areas. The functions provided by database requests, communications pipelines, remote procedure calls, and graphical user interfaces do overlap, and it is reasonable to assume that remote procedure calls may absorb the communications pipeline function as a subset of their capabilities.

SQL is certainly the standard for database access, but since most vendors offer supersets of SQL functions, the standard is a weak one. Work to update the SQL standard to approach the level of function commonly provided by database products is already in progress.

Communications pipelines can be supported by Sun RPC, Apollo NCS, DECnet objects, or IBM APPC. DECnet objects and APPC will dominate sites with large single-vendor commitments to DEC or IBM. Multi-vendor and UNIX sites will settle on one of the remote procedure call methods. Sun RPC is currently implemented on more platforms (primarily in support of NFS), but NCS has a richer set of features and is gaining momentum. IBM's choice for connectivity with their RISC-based UNIX workstations will also have a

bearing on the RPC standards. APPC will probably coexist with other approaches in mixed environments.

X Windows will be the standard upon which GUIs are based; Motif will dominate at the workstation level, but proprietary interfaces will remain on the desktop (Macintosh, PM, NeXTStep, etc). X will be able to coexist with proprietary GUIs because X Windows server software is available for those platforms already.

ATA will continue to track client-server technology and standards. As those standards mature and products that implement them are developed, ATA will evaluate the products, conduct technology pilot programs using the products, and issue guidelines for those products that enhance the SRS computing architecture.

DATE
FILMED
4/21/92