Current Technology Review

Client-Server Computing

TIME-SHARED COMPUTING

Two decades ago, the data processing industry was using powerful mainframes with the users sharing CPU cycles and data storage facilities. Access to the mainframes was tightly controlled by MIS departments and the only method of accessing the data from mainframes was through punched cards or primitive terminals. We saw evolution of pioneering data retrieval and analysis methodologies in the years to come; yet the techniques were arduous and constrained due to centralized resources and user-hostile interfaces. Thus, when PCs came to the computing industry in the early 1980s, their growth was phenomenal, as they provided limited CPU cycles and data storage facilities on user desktops, user control over desktops, and user-friendly PC-based software.

Alok Sinha

PCs, however, were still incapable of handling data processing needs of most large businesses. Local area networks (LANs) provided the solution by connecting PCs and mainframes. Thus, the PC where present, was used as a user-friendly "terminal" to the host. At the same time, we saw the evolution of LAN and PC-based file and print Servers (e.g., Novell Netware started in 1983 and Microsoft MS-Net-based LAN software, such as IBM PC-LAN started in the 1984–85 year). In 1989, in a survey by Infonetics of Santa Clara, 20% of PC LAN buyers cited the desire to share printers, while 22% cited the sharing of large mass storage devices as buying motivation [6]. As we will discuss in the next section, file and printer sharing does not lead to Client-Server applications.

Today, both PCs and LANs have a large installed base in corporations[1]; so have PC-based word-processors, spreadsheets, and database programs. Thus, while the PC-based software has allowed a variety of processing to be done on the desktop, PC LANs have allowed sharing of files and peripherals to take place at the department level in corporations.

We have also seen Unix workstations getting larger installed bases in corporations. These are typically used for "specialized applications." Worldwide Unix workstation revenue from "the commercial market" is expected to rise from $0.8 billion in 1990 to $4.6 billion in 1994, while the revenue from "the technical market" is expected to grow from $6.6 billion in 1990 to $10.7 billion in 1994 [20].

Most of the "Mission critical applications" in corporations, however, have remained mainframe- (and minicomputer-) centric. "Mission critical applications" can be loosely defined to be those information processing and analysis applications whose output is used by corporations for strategic decisions. Almost universally, finance management systems are considered "mission critical applications." The "criticality" of one set of applications over another is often determined by the business of the corporation (e.g., transaction processing systems are often the most "critical" applications for the airline companies).

Indeed, today the Network industry in general and the LAN and Database Server Industry in particular is very excited about "downsizing" mainframe applications to PC Servers and LANs using the Client-Server Computing paradigm. Their enthusiasm is causing MIS shops to take more serious notice of this new paradigm. Of course, this has resulted in the development of new network operating systems and database architectures to support the Client-Server Computing model.[2]

Client-Server Computing Paradigm

Definitions

In the Client-Server computing paradigm, one or more Clients and one or more Servers, along with the underlying operating system and interprocess communication systems, form a composite system allowing distributed computation, analysis, and presentation. We will call such a composite system a "Client-Server System" or simply CSS. In such a system, a Client is a process which interacts with the user and has the following characteristics:

[A] It presents the User Interface (UI). This interface is the sole means of garnering user queries or directions for purposes of data retrieval and analysis, as well as the means of presenting the results of one or more queries or commands. Typically, the Client presents a Graphical User Interface (GUI) to the user (e.g., Microsoft Windows-based interfaces).

As a CSS can consist of multiple Clients, multiple User Interfaces may exist in a CSS, but each Client will have a single consistent UI, e.g., a CSS may have both Microsoft Windows or Presentation Manager (PM) Clients. A CSS may have interfaces in addition to the User Interface for administrative control and system management.

[B] It forms one or more queries or commands in a predefined language for presentation to the Server. The Client and the Server may use a standard-based language such as SQL or a proprietary language known within the CSS. Each user query or command need not necessarily map a query to the Server from the Client.

A Client may use caching and optimization techniques to reduce queries to the Server or perform security and access control checks. A Client may also check integrity of queries or commands requested by the user. Sometimes it may not be necessary to send a query to the Server at all. In such cases, Client may itself perform the data processing requested by the user and satisfy the user query or command. It is however, recommended that Client applications do not provide these functionalities and we ask that they foist these on the Server application. For example, if the Server manages security, it is much more difficult for intruders to break in.

[C] It communicates to the Server via a given Interprocess communication methodology and transmits the queries or commands to the Server. An ideal Client completely hides the underlying communication methodology from the user.

[D] It performs data analysis on the query or command results sent from the Server and subsequently

[1]PC-installed bases in U.S. government, businesses, and education were estimated to be 33.4 million units in 1990 and are expected to grow to 69 million units by 1994. 34.2% of PC-installed bases in 1990 are connected through LANs; 65.2% of PC-installed bases are expected to be connected by LANs in 1994. DataQuest (Mar. 1991).

[2]Network operating systems (NOS) provide network functionalities besides normal (local) file-system functionalities. These typically have network-aware modules embedded within the operating system kernel. Microsoft Lan Manager Server or 4.3 BSD Unix is a very good example of a NOS.

presents them to the user. The nature and extent of processing on the Client may vary from one CSS to another.

The Client characteristics [B] and [D] set it apart from dumb terminals connected to a Host, since it possesses intelligence and processing capability.[3] On the other hand, Client characteristic [D] must not be confused with a PC connected to a LAN, which downloads all the necessary files from a Server or a Host and does all the processing locally.

In a CSS, a Server is a process, or a set of processes all of which must exist on one machine which provides a service to one or more Clients. It has the following characteristics:

[A]. A Server provides a service to the Client. The nature and extent of the service is defined by the business goal of the CSS itself. Thus, an accounting CSS Server provides an accounting data retrieval and processing service to the Client.

A service provided by a Server may require minimal Server-based computation (e.g., print servers or file Servers) to intensive computations (e.g., database servers or Image-Processing servers).

[B]. A Server merely responds to the queries or commands from the Clients. Thus, a Server does not initiate a conversation with any Client. It merely acts either as a repository of data (e.g., file Server) or knowledge (e.g., database Server) or as a service provider (e.g., print Server).

[C]. An ideal Server hides the entire composite Client-Server system from the Client and the user. A Client communicating with a Server should be completely unaware of the Server plaform (hardware and software), as well as the communication technology (hardware and software). For example, a DOS-based Client should be able to communicate with a Unix or OS/2-based Server in the same manner, regardless of the operating system on the Server(s) and LAN technology connecting the Client to the Server(s).

It is advisable, and desirable, that in a multiserver environment, the Servers communicate with one another to provide a service to the Client without its knowledge of the existence of multiple Servers or intra-server communication. Thus, in such a distributed processing environment, the Client should be unaware of the locale of one or more Servers servicing the Client query or command.

Thus, a Client/Server architecture divides an application into separate processes operating on separate machines connected over a network, thus forming a "loosely coupled" system.[4] An application designer divides the user-defined task into subtasks to be completed either by the Client or by the Server(s) within the constraints posed by business goal of the CSS itself and funtionalities provided by the underlying network operating system. The more advanced the network operating system, the smaller (code-wise) the application will be. For example, Microsoft LAN Manager provides a rich set of functionalities geared toward developing CSS. Thus, a CSS running on top of LAN Manager will itself contain less code—which leads to reduced development time. If, however, the same CSS were to run on top on a network operating system which merely provides file/printer sharing, CSS code can easily double.

Industry Perspective

At present, most corporate applications are either mainframe-centric or PC-server-centric. In mainframe-centric environments, users interact with applications running on mainframes through terminals or PC-based terminal-emulators which have the following characteristics:

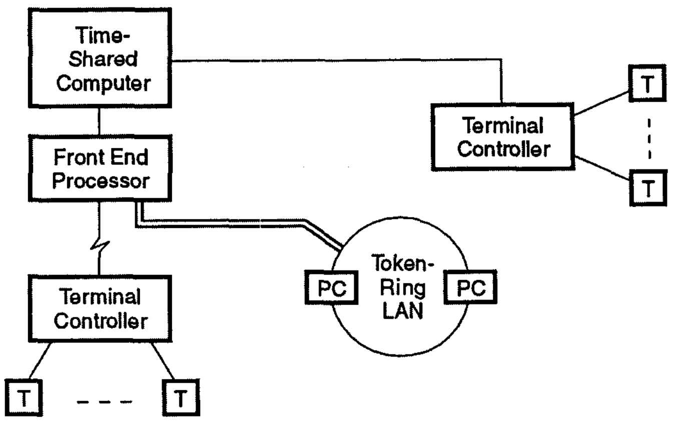
- They may present a proprietary User Interface. Typical terminals or terminal-emulators present a non-GUI interface (e.g., IBM 3270 terminals or emulators). In recent years, the terminal-emulator industry has attempted to present graphical interfaces while hiding non-graphical and proprietary interfaces from the users.
- Typically, all user key strokes and cursor positions are transmitted to the mainframe. Thus, no local intelligence or processing is required, except that required for the transmission of the user commands or queries.
- Simple terminals are typically hardwired to the mainframe or to a local terminal controller, which in turn is connected to mainframes via cluster controllers, while PC-based terminal emulators are either remotely connected to the mainframes through modems or connected to the mainframe through a LAN (Figure 1)
- Typically, all results returned from the mainframes are in terms of cursor positions and characters or strings to be displayed at certain positions on the screen. All computations as well as UI control and rendering is done by the mainframe. This causes excessive loading on expensive mainframe resources, such as storage systems and CPU cycles, and is the main disadvantage of mainframe-centric systems.
- Mainframe-centric systems accord tight administrative control as well as comprehensive system management and performance management facilities.

In PC-Server-centric environments, on the other hand, PCs share applications, and data reside on one or more PC-based Servers. This environment provides flexibility to the individual user, but administrative control and system management tools are minimal. These systems have the following characteristics:

- Typically, the PC-based Server is used to share printers and share common applications and data

[3]See section on X-Windows for differentiation between CSS and X-Windows Client/Server.

[4]Ideally, the Client and the Server may exist on the same machine without any network involvement; nevertheless, almost all Client-Server Systems of interest have a network subsystem.



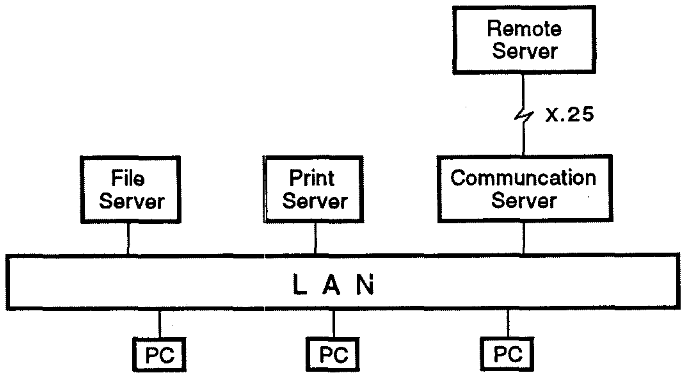


Figure 1. Terminals connected to a time-shared computer

Figure 2. PCs connected to PC-Servers over LAN

(files only). The file Servers provide a range of services to share data among one or more PC applications (Figure 2).

• Each application presents UI and has complete control of the interface as well as rendering of the results. Recent years have seen an explosive growth of GUI-based applications.

• Usually, all user commands or queries are processed on the PC itself. Thus, there must be a large RAM in the PC to enable it to run sophisticated (and thus large) applications. This works against corporate desire to provide cost-effective desktops to the users. Furthermore, the PC-based application may interact with the file Server for accessing (shared or private) data. This causes high volume network traffic since data file(s) must be transported from the file Server onto the PC's local memory. This is particularly true for PC-network-based database programs. Together, these two drawbacks of PC-based applications form the main disadvantage of PC-server-centric systems.

A CSS provides an ideal solution to the drawbacks of both mainframe-centric and PC-Server-centric systems. The most important features of a CSS are

• Desktop intelligence, since the Client is responsible for UI. It transforms the user queries or commands to a predefined language understood by the Server and presents the results returned from the Server to the user.

• Sharing the Server resources (e.g., CPU cycles, data storage) most optimally. A Client may request the Server to do intensive computation (e.g., image processing) or run large applications on the Server (e.g., database servers) and simply return the results to the Client.

• Optimal network utilization as the Clients communicate with the Server through a predefined language (e.g., SQL) and the Server simply returns the results of the command as opposed to returning all the data files.

• Providing an abstraction layer on the underlying operating systems and communication systems such as LAN, allowing easy maintenance and portability of the applications for years to come.

Comparisions with Time-Shared Systems. It is widely acknowledged in the computer industry that CSS systems are being widely considered by corporations for new applications. Often, MIS professionals are confronted with the question "Is Client/Server computing cheaper?" McCarthy et al. [3] present the costs involved in a CSS system and compare it to the typical costs of a mainframe-centric (or time-shared system centric) application. They have broken down the life cycle cost of a CSS into five categories and forecast the cost of developing an application based on the Client/Server computing model, as opposed to the time-sharing model shown in Table 1.

Thus, they summarize:

• Time-sharing has the edge in application development cost through 1992 due to (1) incomplete and incompatible development tools in Client-Server environment and (2) additional complexity of building and testing split applications.

• CSS holds a hardware cost advantage over minicomputers.

• System administration is more difficult in a CSS, since (1) the management tools are immature or missing, (2) a CSS composite consists of multivendor-supplied components, (3) Time-sharing systems allow central administration.

- Software maintenance in a CSS is expected to be lower than in a time-sharing system due to (1) greater reliance on packages as opposed to in-house developments, (2) utilization of new software technology such as object-oriented programming.[5]
- Adoption of the graphical user interface in a CSS will help cut user training costs by as much as 30 to 40%.

Technology

The major components of a Client-Server System are:

- LAN: This is the backbone of the communication subsystem of a CSS and provides low-level communication mechanisms to the network applications.
- LAN-based PC Servers: The Server can be either a packaged product (e.g., file Server, database Server) or a custom-designed Server to meet the needs of the business.
- Mainframe connectivity via PC Server, if desired: This gives the Clients easy access to the vast resources of the existing mainframes. This feature is crucial to the success of a CSS, since it provides a natural migration path to the users and applications downsized from the mainframes.
- Higher-level connectivity support (e.g., RPC) and Client-Server dialogue (e.g., SQL) support
- GUI.

While the Client-Server paradigm itself is not new to the computer industry, it is only now that most of the components of a CSS are commercially available. With the emergence of industry standards in user interfaces such as Microsoft Windows, OSF Motif, IBM Presentation Manager, the development of the Client software has become relatively simple and less time-consuming; these new interfaces, however, have caused escalation of developer training costs.

[5]provided object-oriented tools are available

Similarly, LAN software vendors have provided higher-level communication support. This has been a catalyst in rapid development of the Client-Server systems. The high-level interfaces present a less daunting task of developing and testing the communication subsystems, and allow simpler designs.

Connectivity Interfaces

Low-Level Methods. Traditionally, the emphasis of most early LAN software systems (e.g., Microsoft MS-Net) was on providing such services as file-Sharing and printing. Thus they provided limited and low-level programming interfaces for the LAN system. For backward compatibility reasons, these interfaces are present in the most evolved (or advanced) versions of the corresponding LAN software systems.

Most applications which used (or use) these methods have long development and testing cycles. Software maintenance and portability of these applications are arduous at best. Furthermore, these necessitate business houses to train their programmers to be specialized network programmers. Due to low overhead, however, associated with the transport mechanism itself, high data transmission throughput can be achieved in each case.

Network Basic Input/Output System (NetBIOS). This was originally developed by Sytek, Inc. as a "high-level" programming interface to the IBM PC Network on a broadband adaptor card in August 1984. It was located on the IBM PC Network LAN Adaptor (LANA) as an "extention" to BIOS ROM. Soon, it became a *de facto* session layer standard for the PC LAN industry. In the 1984–85 year, Microsoft developed Microsoft Network (MSNet) software for IBM based on NetBIOS, which formed the basis of a wide array of LAN software systems. Most noticeable were IBM PCLP, original versions of DEC Pathworks and UB Net/One, and 3com 3+Share.

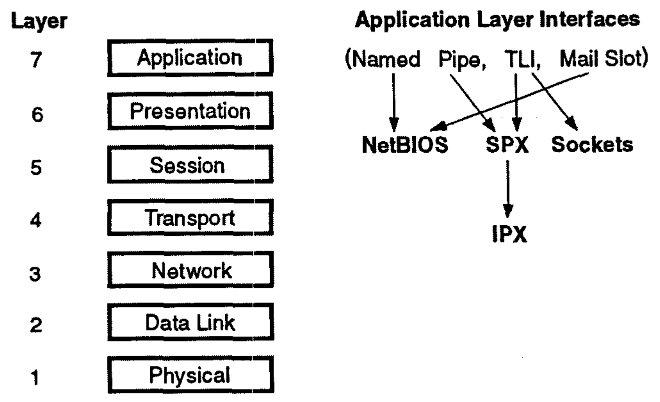
NetBIOS represents a programming interface at the Session Layer as per the OSI model[6] (Figure 3). The programming interface and some implementation specifications have been defined by IBM [9], leaving implementation details to each vendor. Currently, NetBIOS support is provided by all major LAN vendors (e.g., Novell, Microsoft, Banyan) on DOS, Windows, OS/2, and Unix platforms.

Applications interact with NetBIOS support through a data structure called Network Control Block (NCB) (see Figure 4); An application must specify values for *Command* field and may specify values for zero or more fields in the NCB data structure, depending on the NCB Command. Finally, an application "submits" a NCB to the NetBIOS support. The exact interface varies with operating system e.g., while in the DOS environment, an application can call function INT

[6]See Tanenbaum, A. S. *Computer Networks*, Prentice-Hall, Englewood Cliffs, N.J. 07632 1980, for a detailed explanation of the OSI layers.

Table 1.
Relative Cost of Development of a Client/Server System as Compared to a Time-Shared System.

Cost Category Year	Application Development	Hardware Acquisition	Systems Administration	Software Maintenance	Training
1990	+ 30%	− 20%	+ 30%	0%	+ 10%
1992	+ 10%	− 30%	+ 20%	− 10%	0%
1994	− 10%	− 40%	+ 10%	− 25%	+ 10%



The NCB Fields

Offset	Field Name	Length in Bytes	Field Structures
+00	Command	1	□
+01	Return Code	1	□
+02	Local Session Number	1	□
+03	Name Number	1	□
+04	Buffer Address	4	□□□□
+08	Buffer Length	2	□□
+10	Call Length	16	□□□□□□□□□□□□□□□□
+26	Name (Local)	16	□□□□□□□□□□□□□□□□
+42	Receive Time Out	1	□
+43	Send Time Out	1	□
+44	Post Routine Address	4	□□□□
+48	LANA Number	1	□
+49	Command Complete Flag	1	□
+50	Reserved Field	14	□□□□□□□□□□□□□□

Figure 3. OSI model and different interfaces

Figure 4. Network control block (NCB) data structure

5C with the address of NCB in register pair ES:BX to "submit" a NCB; in the OS/2 environment, an application calls NetBiosSubmit() function to "submit" a NCB. In all cases, the underlying NetBIOS support program returns a success or failure message in the *Return Code* field of NCB. Almost all NCB commands can be executed either synchronously or asynchronously.

NetBIOS provides two kinds of data transfer methods—session support and datagram support. Session support provides a reliable data transfer mechanism and flexibility to the applications to establish up to 254 sessions per LAN adaptor per network node (Session number 0 and 255 are reserved). If the data cannot be transferred reliably, an error code is returned to the application program through NCB. Datagram support provides the "best effort" data transfer mechanism, and receipt of data is not guaranteed [9].

Furthermore, NetBIOS provides Name Support to the application programs as well, so that a network node can be addressed—individually by a 16-byte unique name or as a group by a 16-byte group name. Every LAN adaptor has a unique 6-byte permanent name (provided by the adaptor manufacturer), which is appended with 10 bytes of binary zeros to form the first name in the Name Table maintained by every LAN adaptor. Applications can add up to 254 additional names per adaptor per network node. NetBIOS provides a name-resolution protocol to check that a unique name to be added in an adaptor Name Table is not used by any other adaptor on the network and a group name is not present as a unique name in any adaptor on the network [9]. Applications can thus use unique names or group names to communicate with one other.

Internetworking is hidden from applications by underlying NetBIOS drivers or programs and bridges which together provide Source Routing of packets.[7] Simply put, Source Routing is a protocol in which a sender node specifies the route (through LANs and bridges) to the destination node in the frame itself. A route consists of a sequence of route designators, each comprising a LAN number and an adjacent-next-bridge number [16]. The route is carried in the routing information field, distinct from the address field that includes control information. A bridge scans the routing information field for its bridge number sandwiched between the two LANs it joins. The routing information is acquired by the network nodes dynamically with the cooperation of bridges.

A simple Client-Server System consists of a Server which will wait, "listening" (NetBIOS command: NCB_LISTEN) to a 'Call' (NetBIOS command: NCB_CALL) from any Client. Once a 'Listen' matches a 'Call', a session is set up between the Server and the Client. Now, the Server and the Client can transfer data (up to 64KB) bidirectionally through NetBIOS data transfer commands, such as NCB_SEND and NCB_RECEIVE.

[7]A bridge between two local area networks copies some or all packets from one net to another and vice-versa. A bridge operates a data link layer of the OSI model. An intelligent bridge uses intelligent and efficient packet transfer methods instead of transferring all packets from one net to another and vice-versa.

Finally, either end can break the session using NCB_HANGUP command.

NetBIOS provides a solid backward and forward compatible communication method which is a *de facto* standard in PC LANs. It gives an efficient and fast means of transferring data peer-to-peer or Client to Server. However, it causes long product development, testing, and maintenance cycles as most often, it requires low-level debugging using network packet analyzers.

Internetwork Packet Exchange (IPX) and Sequenced Packet Exchange (SPX). These protocols were introduced by Novell in their NetWare LAN products. It is now available in a number of LAN software environments as an added feature for allowing NetWare connectivity and interoperability. IPX protocol is the native communication protocol of NetWare itself and is recommended by Novell as the communication method for Client-Server applications [14]. The IPX protocol is an implementation of the Internetwork Datagram Packet (IDP) Protocol designed by Xerox Corporation. It is essentially a datagram delivery service [15]. On the other hand, SPX protocol is a guaranteed delivery service. Thus, the IPX programming interface exposes an interface at the network layer and SPX programming interface at the session layer (Figure 3).

An IPX packet is identical to a Xerox Internetwork Standard (XNS), which consists of a 30-byte header followed by 0 to 546 bytes of data (Figure 5). A message can be sent to any node on the internetwork by placing it in the data portion of the IPX packet. Each packet must have the destination address, which is composed of the tuple {destination network, node, socket}. The destination network, or destNet in IPX structure, is a 4-byte number specifying a LAN number and is 0 for a node on the same LAN as the sender. The destination node, of destNode in an IPX structure, is a 6-byte number which contains the physical address of the destination node (LAN adaptor) and can be set to 0x FF FF FF FF FF FF for sending a broadcast packet. Finally, a destination socket, or destSocket in IPX header, is a 2-byte field containing the socket address of the destination process.

A SPX packet contains an additional 12 bytes of header to keep track of the packet sequence and acknowledgment number, and may thus contain up to 534 bytes of data. Applications sending SPX packets form SPX connections with destination applications, and SPX retransmits any unacknowledged packets after appropriate timeout intervals. After a certain number of unacknowledged retransmissions, SPX assumes that destination application is no longer listening and breaks the connection [15].

Applications communicate with one another, using either an IPX or SPX programming interface. Prior to using either IPX or SPX functions, applications must prepare and set up a structure called Event Control Block (ECB). Then, they can call the IPX, SPX functions with pointers to ECB, which contains the address of the IPX or SPX packet itself.

A typical Client-Server System consists of a Server that will "open" a socket (Novell IPX API: IPX OpenSocket) and wait to "receive" a packet from any Client (Novell IPX API: IPXReceive). The Server must also "advertise" its presence (Novell Bindery API: AdvertiseService) so that when a Client wishes to connect to a server, it can "seek" a Server (Novell Bindery API: ScanBindery) and determine the Server's address (Novell IPX command: IPXGetLocal Target). Finally, the Client can "send" a request to the Server (Novell IPX API: IPXSendPacket). Novell also provides IPX Setup API, which fills in other necessary fields of the ECB.

IPX provides a fast and efficient communication interface due to little overhead associated with each packet and to its being a simple datagram protocol. SPX provides a stable connection-oriented communication interface which gains in speed due to tight integration with IPX. Both IPX and SPX, however, suffer from the drawback of being low-level programming, similar to the NetBIOS interface.

Sockets. These are the building blocks of communication in the Berkeley Unix system and were first introduced in 4.3BSD as an improvement or generalization of pipes. Pipes provide IPC mechanisms on the same system. Socket is

Figure 5. IPX/SPX packet structure

IPX Packet Structure

```
    unsigned short    checksum;          /* high-low */
    packetLen         packetLen;         /* high-low */
    unsigned char     transportCtl;
    unsigned char     packetType;
    unsigned long     destNet;           /* high-low */
    unsigned char     destNode[6];       /* high-low */
    unsigned short    destSocket;        /* high-low */
    unsigned long     sourceNet;         /* high-low */
    unsigned char     sourceNode[6];     /* high-low */
    unsigned short    sourceSocket;      /* high-low */
```

The SPX packet header consists of an IPX header (30 bytes) and seven additional fields as follows:

```
    unsigned char     connectionCtl;
    unsigned char     dataStreamType;
    unsigned short    sourceConnectID;   /* high-low */
    unsigned short    destConnectID;     /* high-low */
    unsigned short    sequenceNumber;    /* high-low */
    unsigned short    ackNumber;         /* high-low */
    unsigned short    allocNumber;       /* high-low */
```

an endpoint of communication to which a name can be "bound" [11]. Each socket in use has a "type" and has one or more processes attached to it. The "type" of socket determines the method and mode of data transfer in the socket as shown:

- **Stream Socket:** provides bidirectional, reliable, sequenced, and unduplicated flow of data without record boundaries
- **Sequenced Packet Socket:** provides bidirectional, reliable, sequenced, and unduplicated flow of data with record boundaries
- **Datagram Socket:** provides unreliable packet transfer across the socket.
- **Raw Socket:** provides access to underlying communication protocol and is provided so sophisticated Internet applications can be developed

In a sockets environment, the Client (s) as well as the Server must first create a socket by some appropriate socket interface call (4.3BSD call: socket ()). One needs to specify the tuple {domain, type, protocol} when creating a socket. The "type" of a socket has been discussed. One can specify a specific communication domain, e.g., an intra-system domain (in 4.3BSD: AF_Unix), or Internet domain (in 4.3BSD: AF_INET). Furthermore, one can either use the default communication protocol (e.g., TCP/IP) or specify a specific protocol (e.g., "myprotocol").

A socket by itself is nameless, and thus, it cannot be addressed or referenced. In order for Internet communication to take place, there must be an "association" between communicating processes, which is of form {local pathname:local port::foreign pathname:foreign port}. On the other hand, an association {local pathname::foreign pathname} is sufficient for intra-system processes (where 'foreign pathname' means a pathname created by foreign processes, not a pathname on a foreign system) [11]. In order for this association to be built, the Server must call bind interface {4.3BSD call: bind (socket_no, local name, name length)}. Furthermore, the Server must issue a 'listen' call (4.3 BSD call: listen (socket_no, maximum number of outstanding connections)}. Furthermore, it must also make 'accept' or 'select' {4.3BSD call: accept (socket_no,..) or select(...) } calls to listen for Client request to connect; 'select' provides a synchronous input/output request.

The association is completed when a Client makes a 'connect' call {4.3 BSD call: connect (socket_no, server address, size of address)}. The Client must either know the Server address, or it may use socket interface library along with Name Service to obtain the address of the Server. Now, the Client and the Server can transfer data by means of 'read/write' calls {4.3BSD: write(socket_no, buffer, size of buffer) and read(socket_no, buffer, size of buffer)}. Finally, they can eliminate the connection by means of 'close' {4.3 BSD: close(socket_no)} function call.

Socket interface is closer to upper-level (API layer) methods than NetBIOS or IPX/SPX, as each function call provides some degree of abstraction from the underlying transport mechanism. Furthermore, it provides request multiplexing on the Server end and advanced techniques for asynchronous data transfer and out-of-bound data transmission.

Socket interface provides a fast communication mechanism between Unix-based Servers. A number of vendors have provided socket interface not only on Unix workstations, but also on non-Unix desktops such as DOS and OS/2. The programming interface, however, remains low-level and poses the same kinds of problems as NetBIOS and IPX/SPX. The variations among different flavors of Unix provided by various vendors make portability of applications and interoperability between systems a challenging project in itself.

Upper-Level (API Layer) Methods

Typically, this class of communication mechanisms provides an abstraction from the underlying operating system, the networking protocols, and network software and hardware. The abstraction comes from a regulated and carefully designed set of application-programming interfaces (API) which remain unchanged across various platforms (i.e., vendor-independent) within a specific desktop environment (e.g., PC LAN vs. Unix workstation LAN). Porting of applications, based on these APIs, from one platform to another platform becomes an easy task; In some cases, the application itself may not need to even be recompiled. For example, Microsoft Windows-based Named-Pipe applications can be used in the Microsoft LAN Manager or Novell NetWare LAN environment by setting up necessary Dynamic Link Libraries (DLL) and/or Named-Pipe drivers.[8]

While the network software vendors present similar interfaces (APIs) to the applications, they have complete independence in designing and implementing the underpinnings. This leads to efficient data transfer mechanisms, since each vendor has the best knowledge of his or her own system (software or hardware). Furthermore, users of a specific network environment (e.g., Novell Netware) can use any new application software without having to change the environment; They simply need to get upgrade or compatibility kits from the vendor.

Named-Pipes. Named-Pipes were designed at Microsoft as part of early LAN Manager software; they were however, quickly made part of the operating system so that all LAN software vendors can provide support for Named-Pipes. Currently, all major vendors of LAN software provide support for Named-Pipes. The Named-Pipe

[8]It may be necessary to recompile and/or modify the application under certain conditions.

architecture is based on the Client-Server model and all the programming interfaces provided reflect that design guideline. Thus, designing a Client-Server system based on Named-Pipes is easier than designing a Client-Server system based on other methods which can be "adapted" to implement Client-Server communication.

A Named-Pipe is the abstract pipe (Virtual Circuit) that exists between the Server and one or more Clients and provides a reliable unidirectional or bidirectional data transfer channel. A Named-Pipe has a Server end and a Client end, and each must make certain calls to establish a connection. A Named-Pipe can be uniquely identified in global name space by a UNC (Universal Naming Convention) name as shown:

\\<server name>
\PIPE\<pipe name>

where

<server name> is any unique character string (max 16-byte) identifying the Server on the net.
PIPE is a reserver keyword
<pipe name> is a unique character string adhering to the local file system naming convention.

A Named-Pipe has to be "created" {DosMakeNmPipe()} at the Server end of the CSS.[9] While each Named-Pipe must have a unique name, each Named-Pipe can have multiple (up to 255) instances. This feature allows one Server process to create one pipe with multiple instances with each instance servicing requests from a Client. A Named-Pipe can be "created" in the following modes:

- **Wait or No-Wait.** A Named-Pipe in wait mode will cause all reads from and writes to the pipe to be blocked if no data is available. No-wait modes allow all read or write calls to return immediately if no data is available. A Named-Pipe can be changed from blocked to non-blocked (or vice-versa) after a connection has been established.
- **Inbound/OutBound or Duplex.** While an inbound pipe allows data transfer from a Client to the Server, an outbound pipe allows data transfer from a Server to the Client. A duplex pipe allows bidirectional flow of data. The flow of data cannot be changed after a Named-Pipe has been "created."
- **Message or Byte Stream.** A Named-Pipe can provide either a message stream or a byte stream of data. A message stream provides discrete message boundaries and reliable transfer of messages. The stream cannot be changed after a Named-Pipe has been "created."
- **Message or Byte Readmode.** Independent of the stream type of the Named-Pipe, data can be read from a Named-Pipe either as discrete messages or as a byte stream. The readmode of a pipe can be changed after a connection has been established. However, a message stream pipe is read in message readmode and a byte stream pipe is read in byte readmode.

After "creating" a Named-Pipe, the Server must wait for one or more clients to connect to each instance of the pipe {DosConnectNmPipe()}. At this time, a Client can "open" a Named-Pipe by providing the full UNC name of the pipe to a generic file-opening call, e.g., OS/2 call: DosOpen(), Windows call: OpenFile(). This establishes the connection between the Server and the Client. Now, the Client can use normal file-system "read file or write file" calls to read from or write to the file. Finally, the Client can call file-system "close file" to close its end of the pipe. The Server must make a "disconnect" { DosDisConnect()} call when it is notified that the Client has closed its end of the pipe.

Named-Pipe allows the Client end to use normal file-system calls to communicate with the Server, thus making the Client application almost "network unaware"; Thus, the Client end of a CSS can simply focus on UI and data presentation aspects of the CSS. Designing and implementing the Server end of the CSS is made simple due to the Named-Pipe APIs.

Mailslots. Mailslots are part of the Microsoft LAN Manager and adaptations of Microsoft LAN Manager (e.g., IBM LAN Server, AT&T StarGroup LAN Manager, Digital Pathworks). Mailslots provide a simple, reliable or unreliable, datagram (connectionless) channel as well as a consistent application programming interface across platforms and vendors.

"First class" Mailslot message can be used in a CSS for reliable Client-to-Server communication method while "Second Class" Mailslot message can be used for a peer-to-peer communication method. In a CSS, second-class messages may be used by the Server to broadcast messages to all Clients, while first-class messages can be used by the Clients to quickly send some data to the Server. Although not as comprehensive as Named-Pipes, Mailslots are still useful in that they provide the layer of abstraction over the lower-level methods such as NetBIOS or IPX/SPX.

In order to receive messages in Mailslots, a node (Server or Client) must first "make" a mailslot { LAN Manager API: DosMakeMailslot() }; Only the Server running on OS/2 or Unix platform can receive first-class messages. While creating a Mailslot, one must specify the name of a Mailslot as follows:

\MAILSLOT\<mailslot name>

where

MAILSLOT is a reserved key word
<mailslot name> is a character string adhering to file-system naming conventions.

A sender of messages must specify the UNC name of the Mailslot as follows:

[9] Currently, only OS/2 and Unix operating systems support "creation" of a Named-Pipe, thus the Server end of a CSS must exist on these platforms. In the near future, this capability will be extended to Microsoft Windows as well.

\\<computer name>
\MAILSLOT\<mailslot name>

where

<computer name> is a unique character string (max 16 bytes) identifying recipient computer.

Any Client { DOS, Windows, or OS/2 } can send a first-class message to the Server using LAN Manager API: DosWriteMailslot(). The same API is used to send second-class messages to any node on the network. A recipient of messages can read the message from Mailslot using LAN Manager API: DosReadMailslot(). Finally, one can delete a Mailslot using LAN Manager API: DosDeleteMailslot().

While the Mailslot transport mechanism may internally open a connection (Virtual-Circuit) between the sender and receiver nodes for transfering one packet in case of first-class messages, it essentially provides a datagram (connectionless) service. Thus, each call to DosWriteMailslot causes a temporary connection to be established between the Client and the Server making this method a less efficient method for transmitting data. From the application programmer's point of view, Mailslots are most useful for occasional broadcasting of data on a network without using low-level methods such as NetBIOS.

TLI. The Transport Layer Interface (TLI) defines the transport interface to the transport provider, and is not a transport provider itself; The user specifies the transport provider to be used for data transfer which could be based on TCP/IP or STREAMS[10] library [8]. TLI interface is most commonly implemented on top of STREAMS-based transport protocol drivers. Although it is the most commonly

[10]STREAMS was incorporated in Unix System V Release 3 to standardize and augment development and usage of character input/out drivers and to support development of communication drivers. It supports the implementation of services ranging from complete networking protocol suites to simple device drivers. See [6] for details.

Named Pipes

The pseudo-code for the Server end of an application is shown in Figure SB1a and for the Client end of the application in Figure SB1b. Figure SB2 shows the life cycle of a named pipe. In the following example, for the sake of simplicity and clarity, we will assume the Server is OS/2-based and the Client is DOS-based. A Client-Server designer, however, might implement the Client in Microsoft Windows or other graphical user interface environment.

As shown in Figure SB1a, the Server creates a blocked mode pipe and waits for the Client to connect by calling DosConnectNmPipe. The Client can be a local process (on multitasking systems) or a remote process. The remote Client must specify the remote Server name. For example, if Server program was running on a Server named \\harley, Client application will need to specify pipe name as \\harley\pipe\acm. The beauty of the named pipes example shown in Figure SB1a and SB1b is that it works among local as well as remote processes just by changing the pipe name, which can be obtained interactively from the user, unlike the scheme shown in the example.

The Client (Figure SB1b) simply opens the pipe as a file using DosOpen and reads from, and writes to, the pipe. Notice that while the server (underlying implementation such as LAN Manager) allocates two buffers for input and output buffer, the client end does not need to set aside buffers for IPC. The client simply closes its end of the pipe by calling DosClose. This causes the server to be notified of the closure of the client end of the pipe.

Once, the client has been serviced, the server disconnects the pipe by calling DosDisConnectNmPipe. It must call DosConnectNmPipe again to put the pipe in listening mode again so that the next client can connect using DosOpen. Before exiting, the server closes the pipe using DosClose call. After a DosClose call, a pipe has to be freshly created by calling DosMakeNmPipe.

FIGURE SB1a.

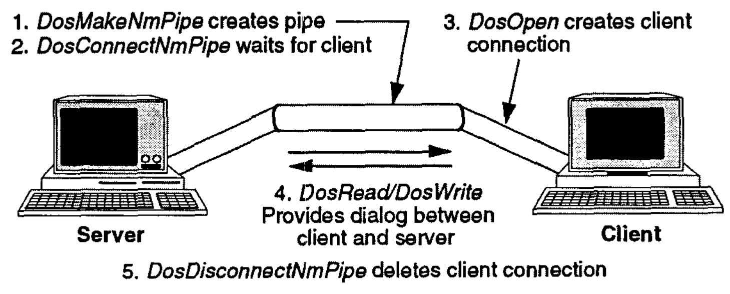
```
#define PipeName      "\\pipe\\acm"
#define IN_BUFF_SIZE          1024
#define OUT_BUFF_SIZE 1024
#define  WAIT_TIME                 -1
int main()
{
        unsigned int    handle, omode, pmode;
        unsigned long  lTimeOut;
        unsigned short status, BytesRead, BytesWritten;
        char           InBuffer     [ IN_BUFF_SIZE ];
        static char        OutBuffer [ OUT_BUFF_SIZE ] = "Hello from Server"
        omode    = PIPE_ACCESS_DUPLEX;           // full duplex.
        pmode    = 0x0001 |                      // 1 instance of pipe
                      PIPE_WAIT                      // blocked pipe
                      PIPE_READMODE_BYTE         // byte read mode
                      PIPE_TYPE_BYTE              // byte type pipe,
        lTimeOut = 10L;                           // 10 milliseconds
        // Create a blocked or wait duplex message mode pipe
        status = DosMakeNmPipe (szPipeName,
                              &handle,
                              omode,
                              pmode,
                              IN_BUFF_SIZE,     //  incoming buffer data size
                              OUT_BUFF_SIZE,  // outgoing buffer data size
                              WAIT_TIME);            // Applications to wait for
                                                       // this time to connect
if (status)
{
      printf('Error creating pipe %d/n', status);
      exit ( status);
}
// The pipe is now made, next we wait/block to connect.
//We will also loop forever waiting for next client to connect
for (;;)
{
      status = DosConnectNmPipe(handle);
      if (status)
      {
            printf('Error in connecting pipe %d\n', status);
            break;
      }
      //We will not get here until the client
      //issues an open. Once we do get here
      //we can do any pipe processing we wish
      //until a close is issued on the pipe.
      do
      {
            // read a request from client
            status = DosRead (handle, InBuffer, IN_BUFF_SIZE, &BytesRead);
            if (status)
            {
                  printf('Error in reading pipe %d\n', status);
                  break;
      }
      else
                  printf('Data received: %s\n', InBuffer);
            // process the request and respond back to client
            status = Dos Write (handle, OutBuffer, strlen(OutBuffer), &BytesWritten);
            if (status)
            {
                  printf('Error in writing pipe %d\n', status);
                  break;
            }
            } while (status ! = 0);
            // Disconnect the pipe. Now, no client can access the pipe
            status = DosDisconnectNmPipe (handle);
            if (status)
            {
                  printf('Error in disconnecting the pipe %d\n', status);
                  break;

            }
      }
      // Close the pipe


            status = DosClose( handle);
            if (status)
                  printf('Error in closing the pipe %d\n', status);
            return (status);
      }
```

FIGURE SB1b.

```
#ifdef LOCAL
#define PipeName "\\pipe\\acm"
!else
#define PipeName "\\remote_server\\pipe\\acm"
!endif
int main( )
{
        unsigned int     handle;
        unsigned short status;
        char        InBuffer     [ IN_BUFF_SIZE ]; ch;
        static char      OutBuffer [ OUT_BUFF_SIZE ] = 'Hello from Client'
        // Open the client end of the pipe in duplex mode
        handle = open(   PipeName,      // Local or remote pipe name
                         O_RDWR |       // Will read and write
                         O_TEXT,        // Text data as opposed to binary
                     S_IWRITE    // ignored   in this case as pipe is not being created.
                     );
        if (handle = =  -1 )
        {
              printf('Error opening pipe [%s) %d\n', Pipename, handle); exit ( 1 ).
        }
        //If a server has already created the pipe, and client has proper
        // access rights, a connection is made between client and server
        //we can do any pipe processing we wish
        //until a close is issued on the pipe.
        do
        {
              // send a request to the server
              status = write (handle, OutBuffer, strlen(OutBuffer);
              if (status = =  -1 )
              {
                    printf('Error in writing pipe %d\n', status);
                    break;
              }
              // receive response from the server
              status = read (handle, InBuffer, IN_BUFF_SIZE) ;
              if (status = =  -1 )
              {
                    printf('Error in reading pipe %d\n', status);
                    break;
              }
              else
                    printf('Data received: %s\n', InBuffer);
              // ask the user if they want to end the program.
              ch = getch();
        } while ((ch ! = ESC) && ( status != -1 ));
        // Close the pipe
        status = close( handle);
        if (status = =  -1)
                    printf('Error in closing the pipe %d\n', status);
                    return (status);
}
```



1. *DosMakeNmPipe* creates pipe
2. *DosConnectNmPipe* waits for client
3. *DosOpen* creates client connection
4. *DosRead/DosWrite* Provides dialog between client and server
Server
Client
5. *DosDisconnectNmPipe* deletes client connection

FIGURE SB2. The life cycle of a named Pipe Source: Ralph Ryan [18]

used interface in the Unix environment, it is made available on non-Unix (e.g., DOS) workstations by some network vendors. Similarly, although the most common transport provider is TCP/IP, any transport layer driver such as SPX can be used in the Novell NetWare environment.

In a CSS, both the Client and the Server must first initiate a dialogue with the provider by calling t_open() function by explicitly naming the provider. Next, both the Client and the Server must associate their logical names to their network name by calling t_bind(). TLI itself does not provide an addressing mechanism. It instead, passes the address in the form of {structure, length of structure} to the transport provider. This allows several possibilities for underlying transport mechanisms, including being TCP or NetBIOS or SPX.

Now, the Server can go into the "listening" state by calling t_listen() and subsequently, becoming ready to accept the Client request by calling t_accept(). At this time, a Client can call t_connect(), specifying the remote Server name to establish a data-transfer channel between the Client and the Server; Applications must first find out the type of service (connection-oriented vs. connection-less) provided by the provider. Once a connection has been established, the Server and the Client can use t_snd() and t_rcv() calls to transfer data units and can finally disconnect, calling t_recrel() and t_sndrel() to gracefully disconnect.

The TLI interface (along with STREAMS I/O system) has allowed Unix applications to be ported across various flavors of Unix and transport drivers. We do not typically see applications written to the TLI interface in non-Unix (DOS, Windows, and OS/2) environments unless they have been ported from Unix; thus it has not impacted the non-Unix world to a large extent. **Remote Procedure Calls (RPC)** provides the framework for a "loosely coupled" distributed processing system in which the processing load may be spread across heterogeneous processors loosely coupled by networks. RPC provides the mechanisms by which a local procedure may be executed remotely on a remote processor and the results returned to the local procedure without the application's knowledge [2, 12].

In a CSS based on RPC, when a remote procedure in the Client end is called, the arguments of the procedure are packaged in a packet and sent to the Server end "magically" and (in simple cases) the calling procedure is blocked. At the Server end, the packet is unpackaged on delivery and a procedure local to the Server executes with the arguments passed by the Client end. At the completion of this procedure, all results are repackaged in the packet, sent back to the Client end, where they are unpackaged from the packet into return values from the remote call procedure. At this time, the calling procedure continues execution.

The RPC implementations can be broadly classified into two major categories

- **ONC[11] from Sun.** In this mechanism, a procedure is executed remotely and its return values are brought back by means of arguments to the call. Figure (6a) illustrates the simplest Sun RPC call, CallRPC(), which takes a set of parameters: (Remote Server name, program name, version, procedure number, External Data Representation (XDR) routine for input arguments, actual arguments, a XDR routine for returned arguments, and returned arguments), where an XDR function provides any data translation required when transferring data between heterogeneous computer systems [12]. In this model, applications need to be aware of underlying network and protocols.

Netwise implementation of RPC for Novell Netware is similar to this mechanism.

[11]Open Network Computing

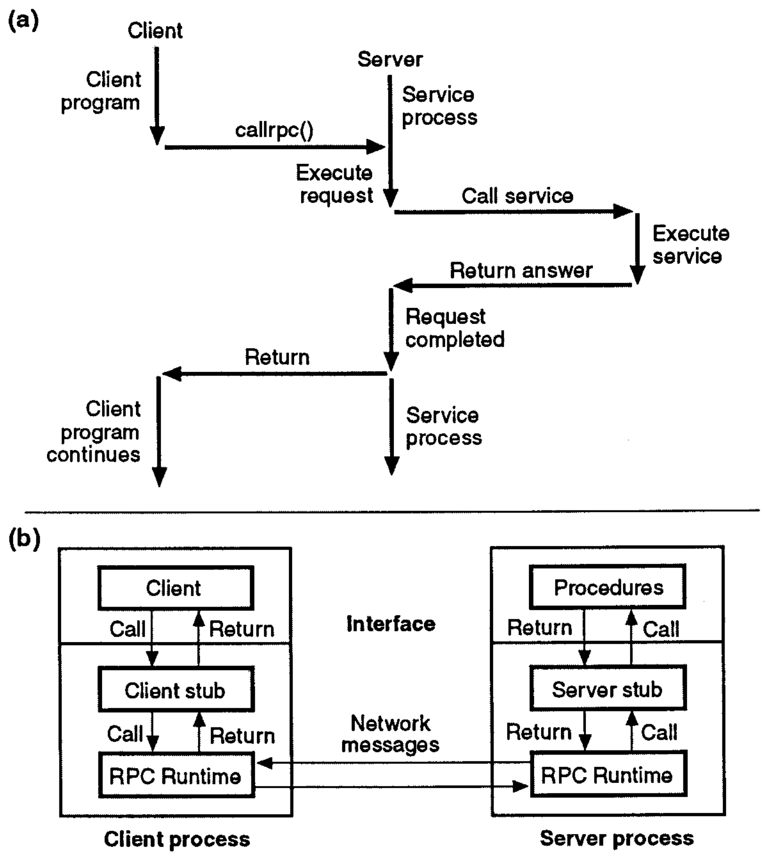


Figure 6. Apollo vs Sun RPC
Source: Byte, 1989 [12]

- **NCA[12] from Apollo.** In this model, a set of routines, called "stubs" are provided at both the Client end and the Server end. Thus, any local calls are routed to the remote Server procedure through the Client and Server stubs as shown in Figure (6b). Typically, the stubs are generated by a RPC compiler which interacts with the RPC runtime library. The RPC runtime library shields the network communication from both the Server and Client applications. For example, Apollo's RPC runtime library provides a connection-oriented data transfer service over the User Datagram Protocol (connectionless) service.

The Distributed Computing Environment (DCE) RPC is based on this model. See [12] for a detailed comparison of the above-mentioned RPC methodologies.

[12]Network Computing Architecture

Advanced Program-to-Program (APPC/LU 6.2) allows peer-to-peer communication between any two programs in a System Network Architecture (SNA) environment, and is the foundation of the "cooperative distributed processing" environment envisioned by IBM.

The entities using APPC communication functions are called Transaction Programs (TP), each of which can have a 64-byte name and an 8-byte identifier; A program can contain one or more TPs. Each TP issues a "verb" to the APPC API interface for some APPC action to take place. A verb is a device, operating system, and underlying SNA component-independent formatted request that APPC executes. Programs use APPC verb sequences to communicate with programs at other locations, while the program itself can be written in any supported language such as COBOL, C, or Assembly [10]. In a SNA environment, each SNA node contains a specific Physical Unit (PU) which manages data buffers and network links. The PU type (2.0, 2.1, 4, and 5) determines the functions provided by the SNA node, such as routing capability, connection capability, and network device addressing capability. On the other hand, a Logical Unit (LU) provides the TPs access to the SNA network by providing a logical abstraction over the underlying PU. Many LUs can be active simultaneously or many TPs can access the services of one LU. Again, the LU type (0, 1, 2, 3, 4, 6.1, 6.2, and 7) determines the functionality provided by the LU. A PU must support LU 6.2 to be able to support APPC-based communication between programs (APPC itself is an implementation of LU 6.2 architecture) [10].

In order for two TPs to communicate, two LUs must first set up a LU-LU 'session.' Sessions control data transmitted, data security, network routing, data loss, and traffic congestion. TPS use conversations to communicate where conversations use LU-LU sessions. A number of conversations can use the same session in serial fashion. Once a conversation is allocated a session, both TPs can transfer data by means of 'receive' and 'send' verbs. (See Figure (7).) Once a conversation ends, the session is deallocated and it can be used by next conversation.

The power of APPC is that any two nodes, which support LU 6.2, can communicate irrespective of their location on the (SNA) net, operating system, or hardware platform; Thus, a DOS workstation can talk to an IBM mainframe, provided both have APPC support software and hardware. Similarly, two DOS machines can communicate with each other via an APPC interface.[13]

APPC can be used as a building block of a CSS. One can envision the CSS Client being the low-end APPC partner (such as a DOS machine) and the Server being the high-end APPC partner (such as a OS/2 Server). The Server can be a mainframe, a minicomputer or a PC-based Server running APPC support software. However, due to physical memory limitations on a typical DOS desktop, it may not be feasible to run APPC support software on the Clients. Thus, some software products, such as the DCA/Microsoft Communication Server, use LAN communication interface (e.g., Named Pipe) on the Client to simply send an APPC verb to a Communication Server, which runs all the necessary APPC support software. The Communication Server uses APPC to communicate with the real Server on the SNA network and return results to the Client through the LAN interface.

The APPC-based Client-Server system is a viable system for businesses which already have an SNA environment and are either downsizing their applications to the PC Servers or want to make mainframe data readily available to the PC-based Clients.

Future Modes of Communication

We anticipate RPC evolving into the mainstream mode of Client-Server communication. Currently, mostly Unix software vendors are migrating toward RPC. In the PC environment, RPC still has to become the key network communication interface. The evolution of RPC will probably coincide with, or be the catalyst for, the evolution of CASE tools for developing Client-Server Systems. It will definitely be the key component of future distributed operating systems and distributed applications.

Database Interfaces

In a Client-Server model-based database computing environment, one or more Clients running on perhaps heterogeneous operating systems and networks make queries to the Server(s). The Server has absolute control over access to data, as well as the responsibility to maintain data integrity and consistency. Therefore, a well-defined interface must be provided to the Clients so they can formulate their queries and send them to the Server, using any communication interface.

In recent years, SQL has become the standard database interface in the relational DBMS. Thus, Clients simply translate one or more user queries into one or more SQL statements, send to the Server, and present the output returned by the Server to the user. On receipt of a SQL query from the Client, the Server checks the syntax of the SQL statements and verifies the existence, access rights of the Client, and data type of each object. At this time, the Server may invoke a SQL query optimizer to find the most optimal path to the referenced data. It finally executes the statement, records the transaction, and returns resulting data, if any, to the Client [5].

Each Client using SQL as the database interface can access any database Server which exposes SQL as the database interface. Thus, a Client end can be implemented in any language or environment. On the other hand, a Client end can be quickly developed using a fourth-generation language. Most advanced software development tools provide network-transparent database functions which convert arguments to the functions into SQL statements at run-time and perform network communication for the application. This allows the developer to focus on database computing aspects of a Client without having to worry about SQL syntax and underlying network software. Acceptance of SQL as standard interface has also been a catalyst in proliferation of off-the-self front ends to the database Servers. Furthermore, most report generation or spreadsheet tools now provide an SQL interface so that users can retrieve information from

[13]There must be a SNA routing node in between the two.

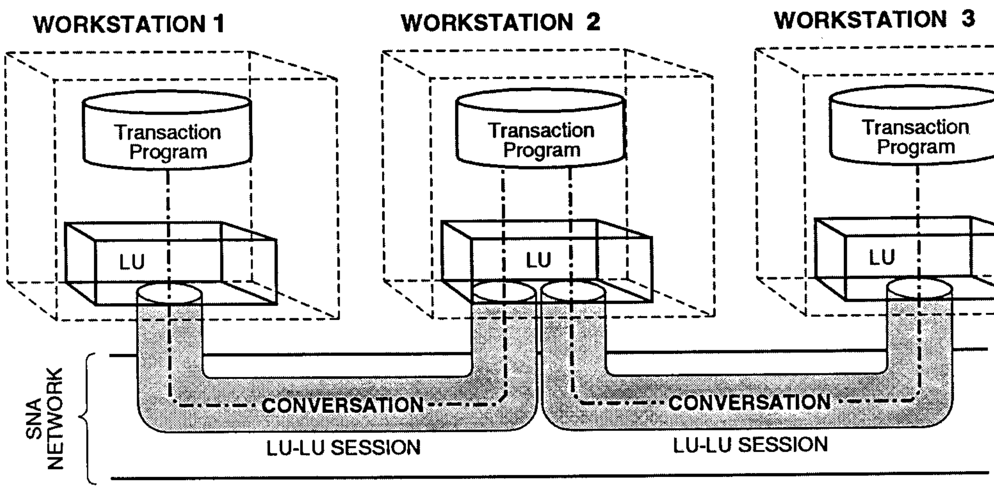


Figure 7. LU-LU session between workstations
Source: [10]

any database Server.

It must be noted that the database interface (e.g., SQL) between the Client and the Server creates network traffic at least an order of magnitude less than typical network traffic generated by PC-based database programs which must bring all the tables stored on the Server onto the PC's local memory.

Graphical Interfaces

Typically the GUI provided with the operating system defines the user interface for the Client end of a CSS. For example, an application running on HP NewWave system uses the GUI provided by NewWave. Currently, the most popular GUIs in the PC computing arena are Microsoft Windows, IBM Presentation Manager, and HP NewWave, to name a few. These interfaces, however, are local to the machine and are not network aware.

In a Unix environment, the X Window System, or the X-Windows, has become the *de facto* graphical interface standard. It was designed and implemented at Massachusetts Institute of Technology (MIT) and has been adopted by most Unix vendors, as well as most Unix workstation hardware and software vendors.[14] The X-Windows is a network-transparent window system that gives applications uniform device-independent text and graphics primitives. X-Windows provides the facilities for generating multifont text and two-dimensional graphics (such as points, lines, arcs, and polygons) in a hierarchy of rectangular windows on a user screen; Every window is a virtual screen which can contain a tree of subwindows which can overlap one another and can be moved, resized, and stacked on top of one another dynamically [19]. Furthermore, the X-Windows provides network transparency so that an application displaying a chart on a screen could be running on a local machine or a remote machine thousands of miles away; Herein lies the difference between the PC environment GUIs and the X-Windows.

The X-Windows facilitates the work involved in the graphical display to be distributed by defining X Server and X Clients (Figure 8). Unlike the Client and the Server of a CSS, the X Server is local to the user's machine, while the X Client can execute on the local machine or a remote system. The X Server and X Client communicate through a well-defined communication protocol called X Protocol, which is the foundation of X-Windows. If both the X Client and X Server are on the same machine, the X Protocol often uses interprocess communication (IPC) mechanisms provided by the underlying operating system. Otherwise, it uses the underlying transport provider to establish a reliable data transfer channel between the X Client and the X Server; X Protocol can communicate all necessary information over a single asynchronous duplex 8-bit bytes stream. X Server sends event notification to the X Client, such as mouse movement or keyboard input. X Client sends requests to the X Server to execute some graphics primitive, such as draw a point { X call: XDrawPoint() }. The X Server does not have to acknowledge any request or even execute any request (Figure 9).

Although X-Windows provides a network-transparent windowing environment, applications need tool kits which provide a higher level of abstraction than accorded by X, such as Draw a Scroll Bar. OSF Motif and Sun Open Look are some of the popular tool kits. Furthermore, one must note that while X (along with added tool kits) can provide one of the key elements of a CSS—namely User Interface—it

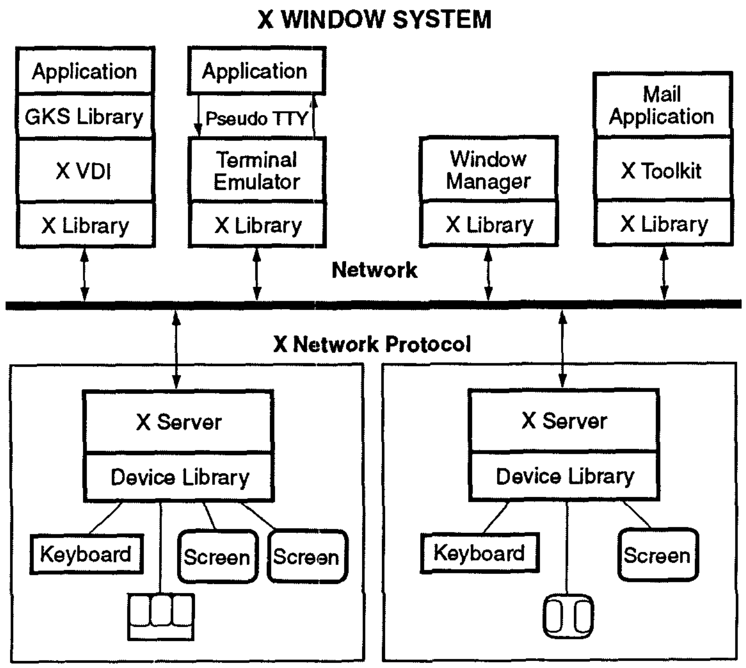
[14]X Window System libraries are now available on DOS and (Microsoft) Windows environments as well.

is not in itself the complete solution. In a simple CSS based on X-Windows, a CSS Client will double both as a X Client and a regular CSS Client. The X Client part will be interacting with the X Server running on a local or remote user desktop for purposes of graphical data presentation (e.g., a chart) and data/query acquisition (e.g., through some GUI form). On the other hand, the regular CSS Client part will be interacting with the CSS Server (e.g., a database Server) for the purpose of finding results of one or more user queries. Thus, one can envision a CSS Client (dubbing as a X Client as well) on a Unix system servicing one or more X Servers running on Unix workstations or DOS-based PCs all connected through a LAN; The CSS Client, in turn, can be interacting with yet another Database Server to service users.

X-Windows is a network-aware graphical interface, where transactions are in the form of requests that are interpreted by an intelligent Server (Host). This is definitely an order of magnitude more efficient than running terminal emulators on the desktop. However, if a X Client is run on a host instead of on the desktop itself, such a system would use host CPU cycles for user interface display on one or more desktops. Thus, it will lead to wastage or expensive host CPU cycles. Furthermore, such a system will still cause high network traffic as compared with, say a Microsoft Windows Client, that just communicates to the Host or Server to receive query results and presents user interface by itself.

X-windows might be a good alternative for a CSS, where most users use Unix workstations and are familiar with Unix and X-Windows. The designer, however, must evaluate simple non-X-Windows implementations using direct network calls to circumvent problems faced with X-Windows-based Clients. Most PC-based LAN software vendors also provide Unix Client software and API interfaces, which can be utilized to develop simple and efficient Clients. For example, the Microsoft LAN Manager for Unix (LM/U) provides almost all of the LAN Manager API interfaces in the Unix environment, which allows for writing efficient Clients. X-Windows Clients on PCs typically suffer from memory problems and slow response time and are, thus, not recommended.

Figure 8. X window Clients and Servers
Source: [19]



Clients

The following are the key issues that a CSS Client must consider:

- **Workstation Operating System (OS).** The workstation operating system (broadly Windows vs. OS/2 vs. Unix) is often decided on by the driving force behind the CSS itself. One must, however, design a CSS so that it can grow to accommodate heterogeneous-OS-based Clients.
- **Hardware Constraints.** The Client should usually be as small as possible so it can be ported to low-end to high-end workstations. This, however, may not always be feasible, especially for advanced applications.
- **Connectivity Constraints.** The underlying connectivity hardware and software itself will influence the design of a Client software. A modular design, however, will ensure that Client application is portable across multiple-vendor-supplied LANs and Wide-Area-Network (WAN) software and hardware.
- **Object-Oriented Design.** In a CSS, object-oriented design techniques can be used to hide the constraints mentioned and make applications portable to new operating systems or LANs.
- **GUI.** The underlying user interface shell (say Windows) is again largely defined by the purpose of the CSS itself and the workstation operating system. In general, the UI should be as modular as possible so the Client can be ported from one operating system to another.
- **Division of Responsibility.** Which processing is done on the Client and which done on the

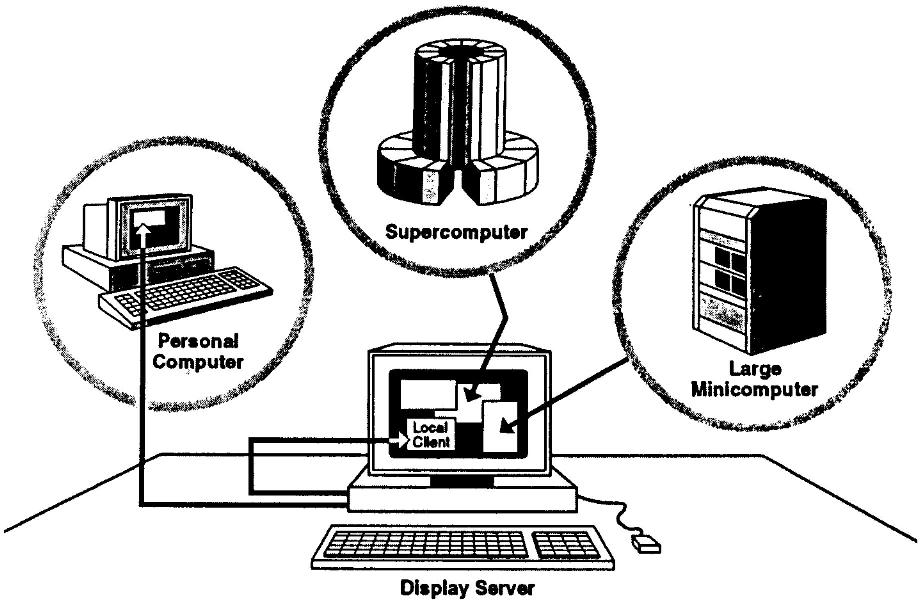
Server? Should the Server be simply a storage facility (e.g., a file Server) that will cause the Client to do the rest of the processing or intelligent information retrieving (e.g., a database Server) in which the Client mostly does presentation and some analysis? These questions can be answered best by the user needs and business goals of the CSS itself.

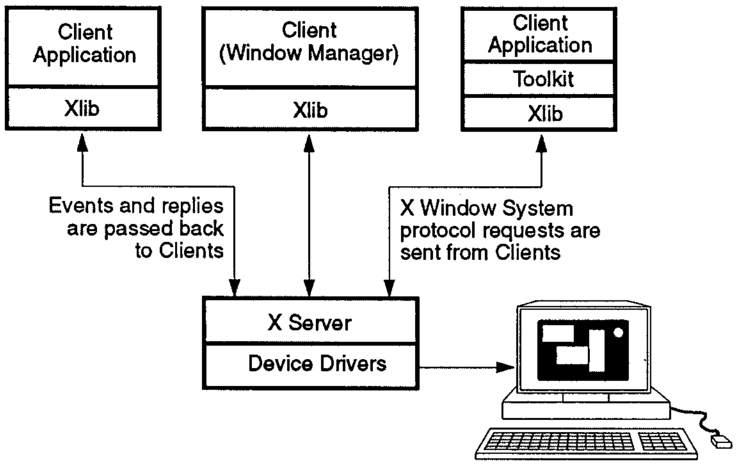
Servers

The importance of proper design of the Server(s) cannot be overemphasized. The following key issues must be considered when designing a Client-Server system:

- **Scalability.** A Server must be scalable to support larger and larger numbers of heterogeneous Clients. For example, in case of database Servers, the design must allow for growth into a distributed database system. A distributed system contains multiple (perhaps heterogeneous) Servers which provide the Clients location transparency,[15] fragmentation transparency,[16] or replication transparency[17, 18] [7].
- **Server Interface.** It is important that the Clients be unaware of any Server features except for the standard interface such as SQL or X-Windows. All Server platform (hardware and software) issues should be hidden from the Clients. This will make the CSS upward-compatible and less susceptible to obsolescence as new innovations in software and hardware unfold.
- **Gateway to Mainframe.** Most medium to large corporations have very valuable and strategic data on mainframes, which the corporate Client wants access to from her desktop. On the other side, the mainframes have large reliable data repository systems that can be used by the corporate Client to store data. Both needs can be easily satisfied by running gateway software on the PC Server and giving all Clients seamless access to mainframe data storage facilities. This method can also be used to gain access to large database systems on mainframes (through database gateway).
- **Disk Space.** When downsizing a large application from mainframes to PC Servers, especially databases, one may experience disk space limitations on a PC Server. While it is

Figure 9. X windows environment
Source: [19]




Client Application
Xlib
Client (Window Manager)
Xlib
Client Application
Toolkit
Xlib
Events and replies are passed back to Clients
X Window System protocol requests are sent from Clients
X Server
Device Drivers

[15]Location transparency makes the user unaware of changes in the physical location of the database.

[16]Replication transparency provides multiple copies of data items across the system, thereby reducing query response time. This, however, causes extra overhead as each change must be replicated to all copies.

[17]Fragmentation and partitioning transparency make the user unware that parts of the database are stored at different locations or an existing (overgrown) database has been split into separate entities.

[18]The current SQL standard cannot support location, fragmentation or replication transparency, and it cannot enforce integrity constraints. The ANSI Standards Committee Group X3H2, as well as SQL Access Group (a vendor consortium) are working on acceptance of updated SQL standards [7].

common to find 600–800MB disk space on the PC Servers, one may obtain Super network Servers which often can be scaled to provide larger and larger disk storage. For example, a Compaq 486/33MHz SystemPro can be configured to provide 1.68GB of disk space.[19] Beyond such limits, the designer should consider splitting the storage (e.g., database tables) over multiple Servers.

• **Security and Access Control.** This has been one of the weak points of PC-based Servers as compared to mainframes. Most LAN software provides password encryption and password-based access control. This may not be sufficient for some strategic applications. In recent years, however, we have seen growth of software and hardware security features or products that one may incorporate into the CSS to make it very secure.

• **Backup, Recovery, and Logging.** This is yet another weak point of PC-based Servers when compared to mainframes. This can be fairly well overcome by integrating multiple software and hardware solutions into the CSS.

• **Fault Tolerance (FT) and Uninterrupted Power Supply (UPS).** FT usually refers to the ability of the Server to withstand the loss of one or more disk drives and disk controllers. UPS refers to making sure the Server gets uninterrupted and steady power during blackouts and brownouts. In this respect, a wide variety of solutions exist which can be easily integrated with the PC-based Server.

• **Performance and System Management.** Current PC-based Servers must grow more to allow complete system management of LANs and Servers from one workstation. Currently, this is one of the major concerns of MIS shops that are planning to downsize to PCs.

• **Internetworking.** Most middle to large corporations that are considering downsizing to PC Servers and LANs are plagued with problems of internetworking. Simply providing a LAN and some form of internetworking solution is often not enough for developing enterprise-wide mission-critical applications. Currently, the lack of effective PC LAN management tools is a critical inhibitor to mission-critical application deployment on LANs [16].

[19]As per Compaq Computer Corporation literature dated 03/30/90

Some Examples of Such Ventures

Microsoft WinSales Project

Chevron Vancouver Project

Chevron Canada used the Client-Server model to revamp their strategic sales-monitoring application that connects Chevron's Canadian distributors to the Chevron's headquarters in Vancouver (Figure 10). The existing application is based on IBM 3090 and is a 20-year-old batch-processing application [21]. It is used for order entry, tax and inventory control, and for producing strategic management reports. Thus, it is a mission-critical application, supporting a combined on-line transaction processing system (OLTP) and decision support system (DSS).

Based on organizational and user needs, they decided the database Server must be multiuser, scalable with support for both OLTP and DSS, and most important, be based on the Client-Server model. As for the Server platform, it must support the database Server selected, be scalable, and must support local area networking as well as wide-area networking. Finally, CSS must support various Clients and end-user tools.

According to Soper, they built a prototype based on Sybase/Microsoft SQL (database) Server on a OS/2 platform. The Windows or OS/2-based Clients were connected to the Server via Microsoft LAN Manager LAN software. The prototype was installed in early 1990 with the full system expected to be commissioned by 1992. At that time, the CSS will be handling about 165,000 transactions per month coming from over 35 remote sites, as well as handling requests from about 65 local Clients. In a fully installed system, each Client would connect to a local Server which would contain data pertinent to the Client's needs. In total, there would be 300 relational tables spread over multiple Servers which are replicated to a central Server database which is estimated to be as large as 3GB.

iLan Project (excerpted from [22])

iLan is a consulting and software development firm based in Columbus, Ohio. They developed an imaging system, based on the Client-Server model, for a customer. The iLan imaging system supports very high-volume scanning and indexing of images for mission-critical applications. For billing, the system can produce a list of invoices from the index, find and print out copies of all of them.

In this System, a front-end processor (a DOS-based workstation) scans in an image, allowing the operator to enter index information or queue up the image for later indexing. Other operators find out what has been queued up when they start work, call up the image and key the indexing information. When indexing is complete, the image is stored on an optical disk, while the indexing information (key words, comments) is written to an Oracle database from Oracle Corp., Redwood City, Calif., on the Server machine.

For this installation, they picked Vines network from Banyan, Westborough, Mass., with Oracle for Vines as the database Server. They decided to use the Unix-based Server, since it allowed them flexibility. Furthermore, Oracle database could be programmed to hold only the index as opposed to the image itself, thus saving the database from storing huge images.

Now, they are porting their application to different Server platforms (e.g., OS/2, AIX) as the Oracle database Server is becoming available on each such Server platform. Thus, by choosing a Server

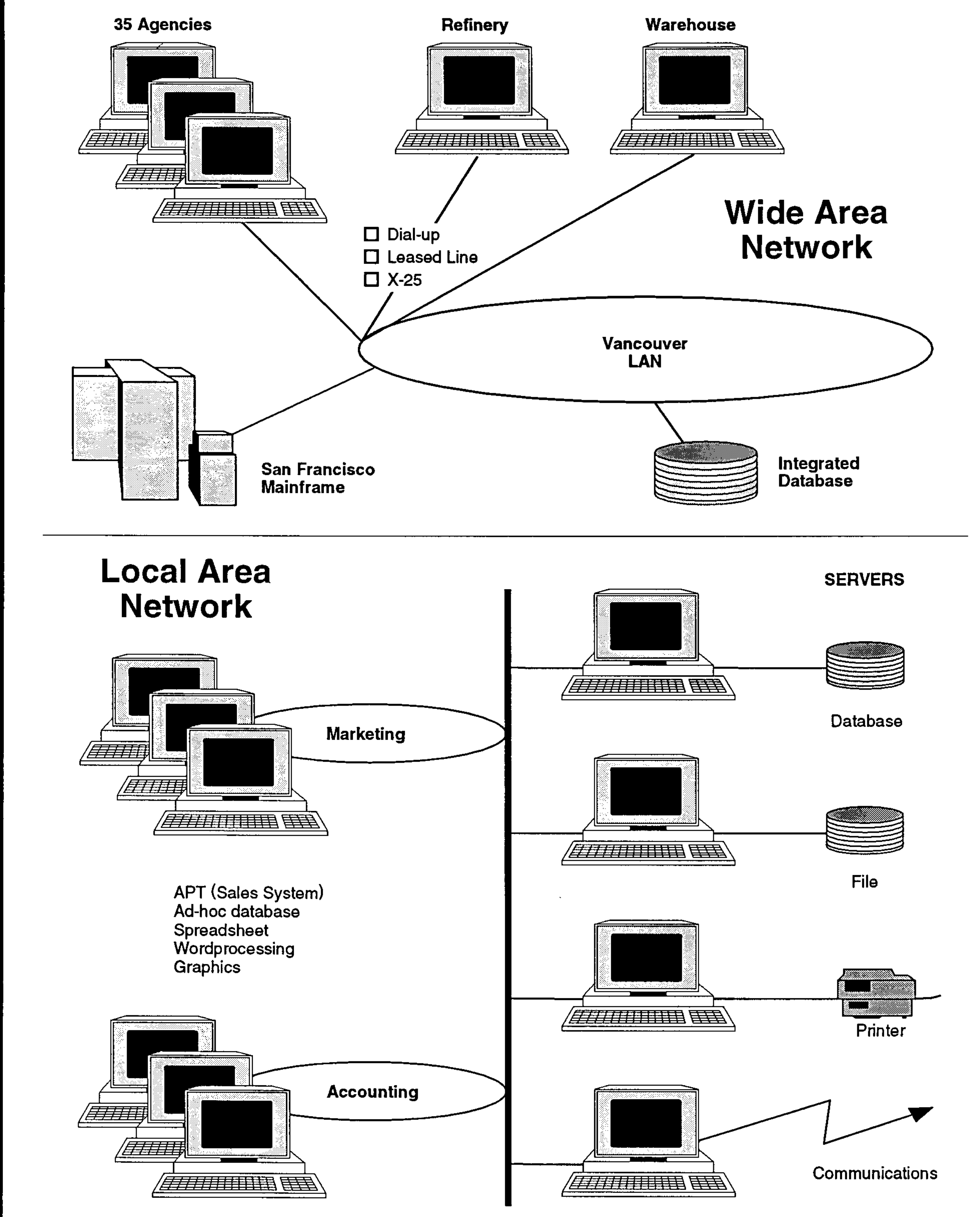


Figure 10. Chevron Vancouver project
Source: [21]

that runs on multiple platforms, iLan retained the flexibility to change the Server hardware and operating system wherever it is beneficial to do so. Their application code does not have to change as the Server platform is changed, since Oracle will maintain a consistent Server interface.

Commonwealth Bank (Australia) Project

Commonwealth Bank of Australia is one of the largest commercial banks with 50,000 employees and 1,800 branches. They are implementing Access to Sales and Service Information System (ASSIST) in their branch offices to provide their customers with faster service and easier access to banking information. The system is based on the Client-Server computing model.

Prior to installing the new system, the bank faced a daunting paper chase. Opening a typical checking account could take up to 20 minutes and involved filling out several forms which had to be processed by one or more bank officers. Furthermore, data entry was yet another problem. Each day about 200 operators entered data into 3,090 IBM mainframes which were again checked for errors. All of this led to slow, yet expensive service.

Thus, the bank decided to implement ASSIST in all the retail branches. So far, they have installed ASSIST in over 250 branches, which led to a 90% reduction in paperwork, and accords a pleasant customer service environment.

Under the new system, each branch has a Token-Ring LAN with a NCR or Nixdorf 386-based Server with 16MB of memory. Each Server has a NCR-adapted OS/2 operating system with Microsoft Lan Manager LAN server on top of the operating system. NCR-adapted Microsoft/DCA Communication is also loaded on a Server to allow an open communication link with central IBM 3090 mainframes. Finally, Microsoft/Sybase SQL Server is also run on each Server as the database server. Each Server supports anywhere from 5 to 10 NCR or Nixdorf 386SX workstations, which have printers, magnetic swipe readers, and smartcard readers as peripherals. The magnetic card swipe allows customers to use a credit card or dept card for identification during transactions.

The user interface consists of a simple series of windows, allowing customer service representatives to check account balances, open new accounts, perform overseas transactions, or access customer information. The user interface hides the network and SQL-based transactions between workstations and the SQL Server. All error-checking, customer validation, and financial data integrity checking has been implemented as triggers in the SQL Server, which gets activated by each transaction. Each 'trigger' is a compiled C/SQL function, which gets called by the SQL Server to enforce user-defined semantic and integrity rules. The Client software and all SQL Server programming was done by the bank's information services department.

By the end of 1992, all of the

Client/Server Architecture of Winsales

FIGURE SB3a. depicts Microsoft MIS Winsales base client/server architecture. The architecture comprises personal computers running Windows[1] applications with SQL Server,[2] a relational database management system, as the primary server application. The client application is an executable that runs on top of the Microsoft Windows operating system. The user controls the client application by initiating keyboard and mouse events. These events generate recognizable Microsoft Windows system messages that are trapped by the windows application and are used to trigger application functions. Application functions in turn call DB-Library functions.[3] DB-Library is the application programmer interface (API) to SQL Server. It is used to establish and break connections with the server and to submit and receive data processing requests, primarily in terms of SQL. DB-Library makes the necessary calls to the Lan Manager[4] API, which in turn provides access to server resources over Ethernet under the XNS[5] protocol. On the SQL Server side, each server application is made of data, tables, indices, and other relational database objects. The server receives requests from the client application via Lan Manager. These requests in turn are submitted to the database engine for processing.

This architecture has several advantages. All knowledge of the network and

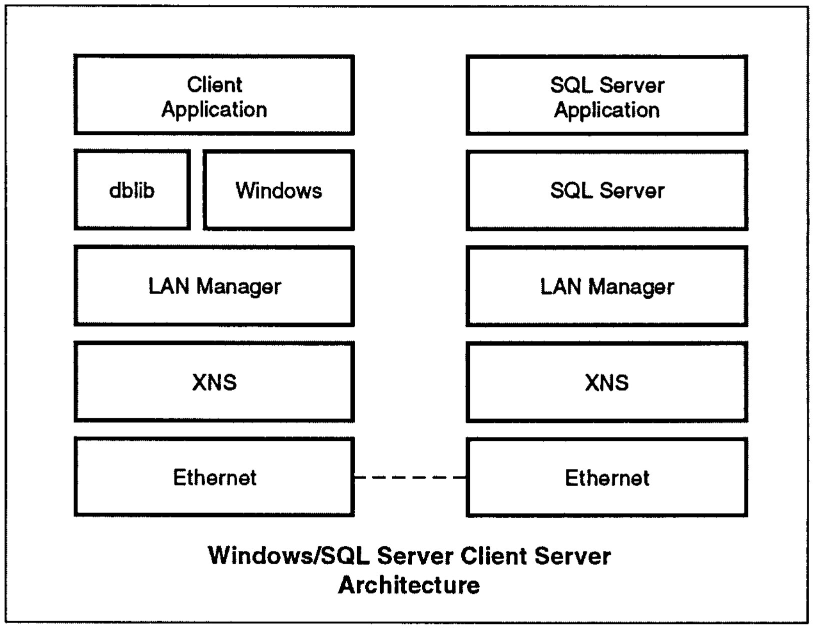


FIGURE SB3a. Windows/SQL Server Client server architecture.
Source: [17]

Commonwealth Bank's 1,800 branches are planning to have similar systems instâlled. The bank also plans to use the Micro Tempus Enterprise Router to develop an enterprise-wide routing system or virtual LAN that connects 1,800 local Servers to the IBM hosts. This will allow monitoring and control of all Servers from a single location.

Microsoft MIS WinSales Project: (excerpted from [17])

The WinSales application was designed at Microsoft MIS house to support the execution, administration, and analysis of separate telemarketing campaigns. In addition, information collected by one campaign should be available for reuse for future campaigns and the same database should support multiple campaigns concurrently. The system is designed to be used by sales representatives from a large number of field offices for telemarketing, as well as customer surveys. (See sidebar.)

server implementation is encapsulated within the client/server API and therefore entirely transparent to client applications. For example, a SQL Server application may be moved to a new hardware platform overnight with no effect on the client application. With the appropriate gateway technology, DB-Library may be used to access databases other than SQL Server, including operating system files such as VAX RMS or RDB files. Several client applications may run concurrently on a single workstation and communicate with each other through Windows. In addition client applications may access multiple and diverse server applications during the same session. In Winsales environment, multiple SQL Servers, corporate email servers, and fax servers are all used to link individual workstations to corporate data, each other and the rest of the world (see Figure SB3b).

[1]Microsoft Windows

[2]Microsoft SQL Server and Sybase SQL Server

[3]part of the SQL Server product

[4]Microsoft Lan Manager

[5]Xerox Network Standard

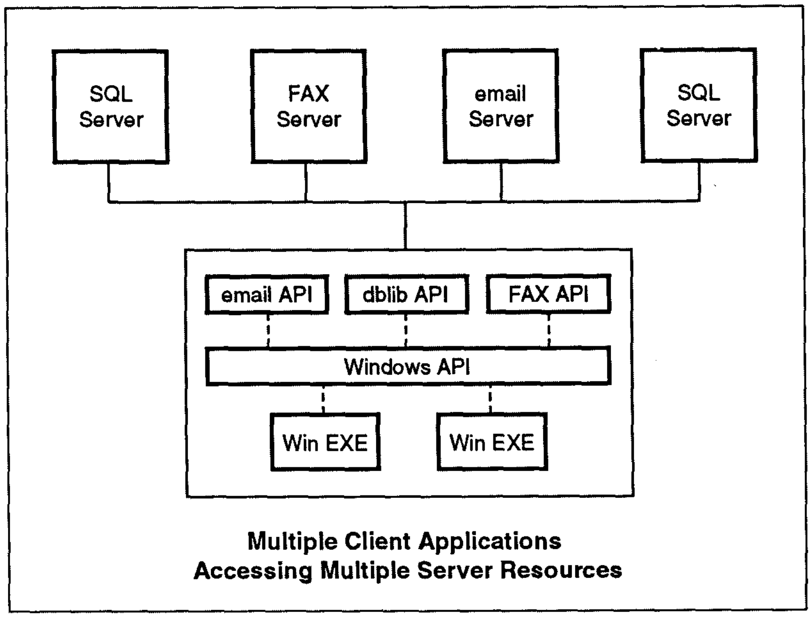


FIGURE SB3b. Multiple Client applications accessing multiple server resources. *Source:* [17]

Conclusions

Today, a wide spectrum of Client-Server Systems are being implemented to make information easily accessible to the end-user. This article attempts to delineate major components of a Client-Server system with special emphasis on communication interfaces. While most Client-Server systems being developed are database computing related (i.e., the Server is a database Server), not all are or will be.

Today, most of the components of a Client-Server system are available commercially. Furthermore, most components, namely LAN software, database Servers, and User Interface software, are being provided by a number of vendors, giving MIS houses freedom of choice. Nevertheless, most MIS houses still have to do some in-house development. Finally, system integration of CSS components is a daunting task which all MIS houses must perform.

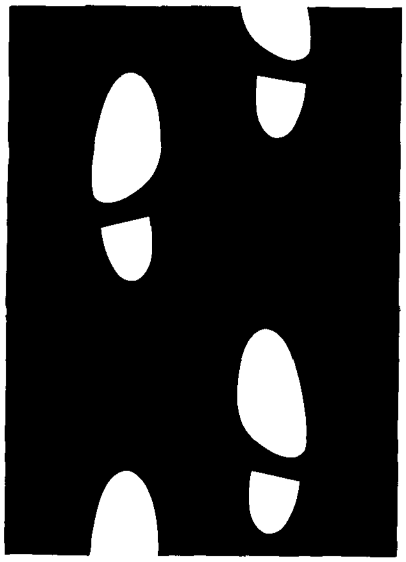
With the growth of Client-Server systems, one can anticipate that most vendors of CSS components will work toward open systems, which are easy to integrate and will require less in-house development to fit in within the CSS. Furthermore, one can foresee evolution of fourth-generation languages (4GL) and CASE tools, which will allow rapid prototyping of Client-Server systems.

Acknowledgment

I would like to thank Ray Patch, Charles Rudd, Brian Valentine, Bob Kruger, Dov Hovel, Steve Dudgeon of Microsoft Corporation and Bill Soper of Chevron Corporation for their comments and/or assistance in putting this article together. ◘

References

1. AT&T. AT&T streams. ISBN 0-13-940537-2025, 1987.
2. Birrell, A.D., and Nelson, B.J. Implementing Remote Procedure Calls. *ACM Trans. Comput. Syst. 2*, 1 (Feb. 1984), 39–59.
3. Business Research Group. PC Lan integration and management: User trends. Newton, Mass., June 1991.
4. Client-Server Computing. Datapro reports on PC communications.



Make Tracks...

... to your nearest mailbox and send for the latest copy of the **free** Consumer Information Catalog.

It lists about 200 free or low-cost government publications on topics like health, nutrition, careers, money management, and federal benefits.

Take a step in the right direction and write today for the free Consumer Information Catalog. Just send your name and address to:

Consumer Information Center
Department MT
Pueblo, Colorado 81009

A public service of this publication and the Consumer Information Center of the U.S. General Services Administration.

Tech. Reps., McGraw-Hill, Inc., Delran, N.J. 713–101. June, 1990.

5. Dale, R. Client-Server database: Architecture of the future. *Database Program. Design.* (Aug. 1990), 28–37.
6. Fields, D., Ed. *Client Server Computing: Making it Work for You.* Parallan Computer, Inc., Mountain View, Calif., 1990.
7. Gold-Bernstein, B. Does Client-Server equal distributed database? *Database Prog. Design* (Sept. 1990).
8. Harris, D. TLI. Unix Networking. S.G. Kochan and P. Wood, Eds., Hayden Books Unix System Library, pp. 171–202.
9. IBM Corporation. IBM Local Area Network Tech. Ref., SC30-3383-2, 1988.
10. IBM. APPC Programming Reference. Operating System/2 Extended Ed., Version 1.3. Sept. 1990, Order no. 01F0295 S01F-0295
11. Leffler, S.J. et al. An advanced 4.3BSD interprocess communication tutorial. Computer Systems Research Group, Department of Electrical Engineering and Computer Science, University of California, Berkeley, 1986.
12. Manson, C. and Thurber, K. Remote control. *Byte* (July 1989) 235–239.
13. McCarthy, J.C., Bluestein, W.M. Is Client-Server cheaper? Forrester Research Inc., Cambridge, Mass. (Aug. 1990).
14. Novell, Inc. Client Server tutorial. Professional Development Series, C Network Compiler/386, Austin, Tex.
15. Novell, Inc. Netware loadable module library reference. Professional Development Series, C Network Compiler/386, Austin, Tex.
16. Pitt, D.A., and Winkler, J.L. Table-free bridging. *IEEE J. Selected Areas in Commun. SAC-5,* 9 (Dec. 1987).
17. Rudd, C., McKee, P., Yip, T. The development of WinSales: A Client/Server application. Microsoft Internal Paper. Dec. 1991.
18. Ryan, R. *Microsoft Lan Manager: A Programmer's Guide.* Microsoft Press, One Microsoft Way, Redmond, Washington, 1999.
19. Scheifler, R.W. et al. X Window System: C Library and Protocol Reference. Digital Press.
20. Sharp, B. *Commercial Success. Lan Comput.* (June, 1991), 24–25.
21. Soper, B. Private communications, Project Manager, Chevron Canada Ltd., Vancouver B.C., Canada.
22. Winston, A. OS/2, Unix competing for server selection. *Softw. Mag.* (May 1991).

CR Categories and Subject Descriptors: A.1 **[General Literature]**: Introductory and Survey; C.2.4 **[Computer Communication Networks]**: Distributed Systems—*distributed applications*; H.3.4. **[Information Systems]**: Systems and Software—*information networks;* K.8.0 **[Personal Computing]**: General—*games*

General Terms: Design, Performance

Additional Key Words and Phrases: Client-Server computing

About the Authors:

ALOK SINHA is a software design engineer at Microsoft Corp. in Redmond, Wash. His current research interests are distributed computing, directory service, and LAN. **Author's Present Address:** Microsoft Corp., One Microsoft Way, Redmond, Wash. 98052-6399; email: aloks@microsoft.com