Application reference designs for distributed systems

by J. J. Shedletsky
J. J. Rofrano

This paper is based on the findings and conclusions of a client/server work group that was commissioned in 1991 to report IBM's technical strategy for client/server computing. Although there are countless variations for designing applications and interconnecting components in a distributed environment, there seems to be a finite number of variations that represent what a large majority of customers want to build. The intent of the work group was to explore the possibility of defining a set of application "reference designs," which would represent the distributed designs that customers are building today or want to build in the near future. This paper documents the customer scenarios, the reference designs that represent them, and the requirements that were generated for the underlying system software. The work group concluded that the reference designs described herein represent our best working assumption about "where customers are going" with distributed application designs. The discussion should give those who have not yet begun to exploit distributed systems a starting point and considerations for their design work.

Customers are increasingly viewing their diverse collections of computing resources as a single enterprise-wide asset. Business imperatives such as global reach, operational efficiency, better customer service, or competitive advantage have led to solutions that require an increasing number of connections between these computing resources. Users may need access to data located in more than one database to accomplish a business task. Different databases may exist on computers from different vendors and in remote locations. New applications must interact with old applications as increasing levels of cross-enterprise integration are required. And users may need to access applications and data from different locations as they move around, or, conversely, applications and data may be moved to geographically dispersed users. Computing solutions in the 1990s will require unprecedented levels of interoperability between both hardware and software products of computer vendors.

In addition, the increased spectrum of computing choices, ranging from mainframes, down through minis, servers, workstations, and personal computers, has introduced new models for constructing these solutions. Encouraged by the lower cost of storage and computing power, customers are seeking to "off-load" CPU cycles and data storage from centralized mainframes.

Terms such as *client/server*, *cooperative processing*, and *open network computing* have been used to characterize alternatives to solutions based on a centralized model. These terms have a variety of meanings to different people, depending on their frame of reference. For the purpose of this paper, they are variations on a more generic design point called *distributed systems.*

These new distributed models are aimed at solutions where the user, the application logic, and the data are no longer constrained to be on the same system, thus permitting the customer more flexibility in selecting platforms suited for a particular mission.

Computer vendors face the challenge of providing solutions that offer the flexibility of a distributed model while at the same time accomplishing the increased cross-enterprise integration required by customers. Although these goals are not totally contradictory, they do tend to counterbalance each other. Distribution carried to a total dispersion of computing resources would diminish the ability to provide cross-enterprise solutions.

Likewise, a concentration of all computing resources into one centralized platform would eliminate the option of utilizing advanced workstations and increase the communication burden of connecting users to that resource. Computer vendors who fail to identify and provide system solutions that strike an appropriate balance will be at a competitive disadvantage in the 1990s.

The objective of this paper is to define a set of application models that represent these emerging solutions and the environment that is necessary to support them.

Our approach examines some leading-edge solutions that do seem to strike an appropriate balance between the considerations of decentralization and cross-system integration. Analyses of these examples identify the kinds of enabling software functions required and suggest the basis for a "building block" architecture of distributed software services.

We use the term distributed system to refer to a distributed collection of users, data, software, and hardware whose purpose is to meet some defined business objective. The design of a distributed system describes how the users, data, and software are placed in relation to one another in a distributed network of more than one hardware platform. The customer designs we have examined exhibit definite structures that come about because of specific requirements. More importantly, we have observed that these structures seem to occur repeatedly, even across applications of different industries such as banking and manufacturing, and across different choices of underlying hardware platforms.

Reference designs

A complete description of a distributed solution design requires three levels of specification:

- Physical network
- System services infrastructure
- Applications software

The lowest layer is the physical network, which describes the hardware platforms and the network connections between them. The system services infrastructure is the collection of system software services installed on the hardware nodes. It provides a distributed execution environment for the applications that support the enterprise-specific business tasks.

The physical network is often dictated by geographical, organizational, or corporate strategic considerations. For example, a global banking enterprise may have several data centers spread across different time zones. A large insurance company or retail bank may have branch offices with a central office. A manufacturing enterprise may provide advanced workstations and servers for engineering work groups, whereas a large retailer may seek to increase staff productivity by providing a workstation on every desk. Large organizations may have separate systems designed to support functional organizations like finance, billing, plant floor, and inventory control. A large hospital or a university may have diverse systems in largely independent departments. Finally, merged enterprises may be forced to deal with installed bases of completely independent computing systems. Each of the considerations in these examples will have a primary effect on the design of the physical network.

Each node in the physical network requires system-level software to support the role it will play in the distributed solution. Most nodes require an operating system and a collection of enabling services such as communications, file systems, database managers, and presentation services. The enabling services may stand alone or be distributed. For example, a file system may support local files only, or it may provide access to both local and remote files. The specific services required depend on the overall design of the dis-

tributed system. It is the purpose of this paper to identify these services.

The application software implements the enterprise-specific business tasks supported by the distributed solution. The application programs installed on each node define the role that node will play in the distributed system. For example, a workstation may play a front-end role to multiple back-end transaction servers. The front-end application would manage a dialog with the user and coordinate requests to back-end transaction applications. The needs of the application programs dictate the enabling services required in the underlying system software. Applications themselves may be broken into parts and distributed or simply rely on underlying distributed system services, such as a distributed file system.

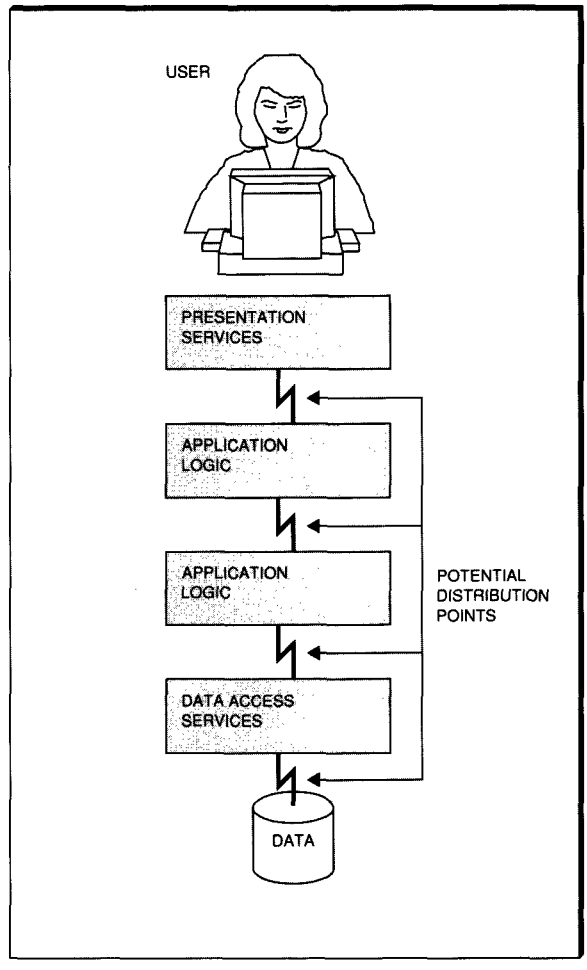
Figure 1 depicts different ways in which an application and the system services it uses may be broken into parts to run on different hardware platforms.[1] Considerations of communications cost, performance, support staff availability, security, and protection of mission-critical data often determine the approach chosen.

With the wide range of choices and considerations, it would seem that there could be a theoretically infinite number of possible designs for a given distributed system. How then, can we hope to gain any insights into the future direction of distributed systems or the services required?

Our approach was to survey a number of real-world solutions being designed or implemented by customers. Approximately 50 solutions were surveyed from enterprises in the banking and finance, securities, insurance, manufacturing, process, retail, transportation, and health care industries. "Leading edge" distributed projects were chosen under the assumption that the pioneering designs of today would be the routine designs of tomorrow. Many of the customers were large enough to develop or contract for their own distributed services if none were available from vendors.

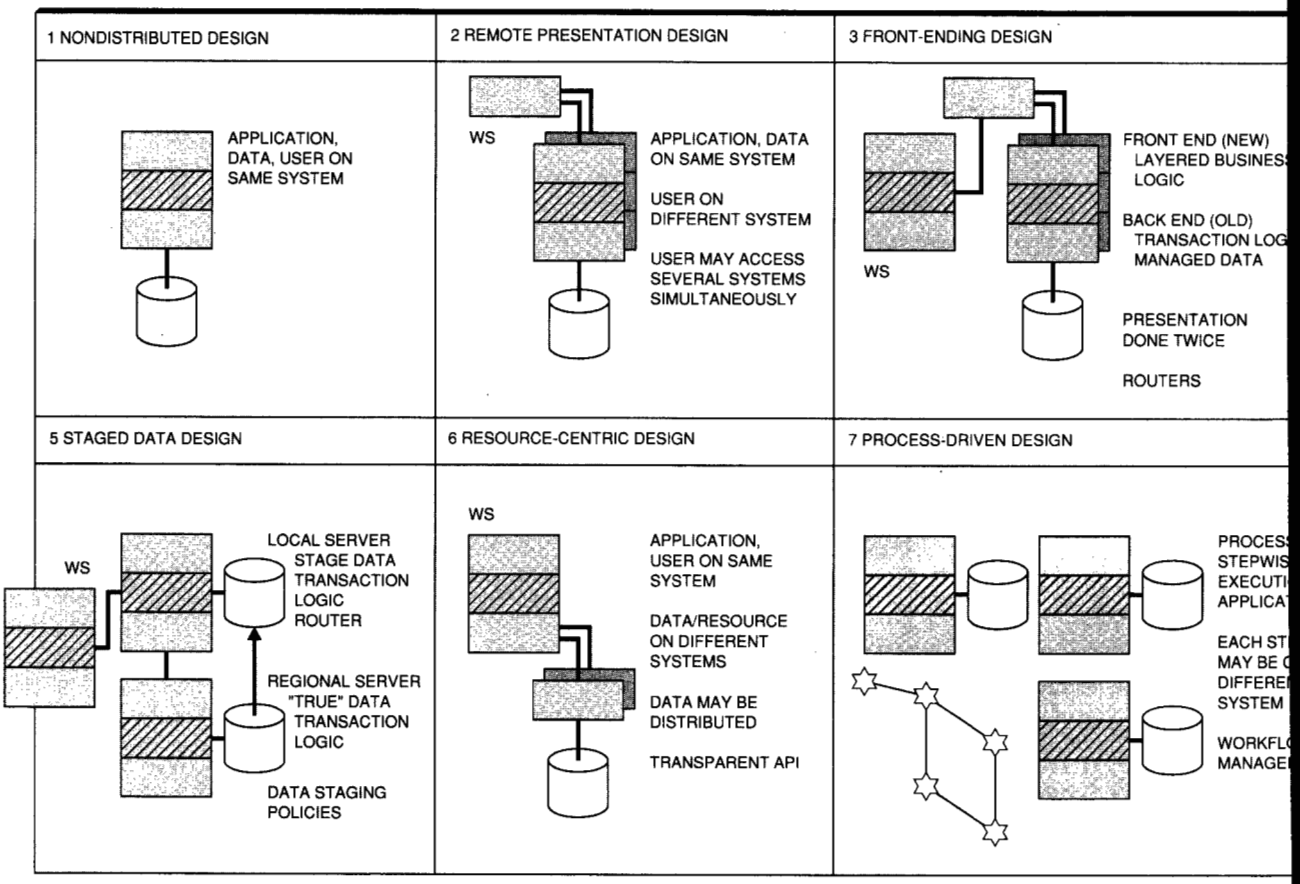
It was observed that the designs chosen for these distributed systems were not infinite in variety. In fact, certain designs seemed to occur repeatedly in the survey, across industries studied, and on varying hardware platforms. It is convenient to categorize these designs into classes of distributed systems that we refer to as reference designs. Figure 2 summarizes the seven classes of distributed systems observed. An eighth class, the nondistributed case, is included for contrast.

Figure 1 Points for distribution



Each reference design in Figure 2 depicts a physical network in which each node plays a defined role. The shaded areas represent application software, and the unshaded areas represent the system software. Stored data are represented by the disk symbol. The reference designs are distinguished by differences in the placement of application software, system software, and data in relation to one another. Descriptive names were chosen to characterize the design approach used in each reference design. The "front-end-

Figure 2 Classes of distributed systems observed



ing" reference design, for example, is used to layer value-added application logic on the workstation in front of existing back-end systems and databases.

The reference designs are described in detail in the following section. The factors and trade-offs customers consider in selecting these designs are also discussed. Included with the description of each reference design is an analysis of the system service infrastructure needed to build it. The required services were identified by observing those in the customer-devised solutions.

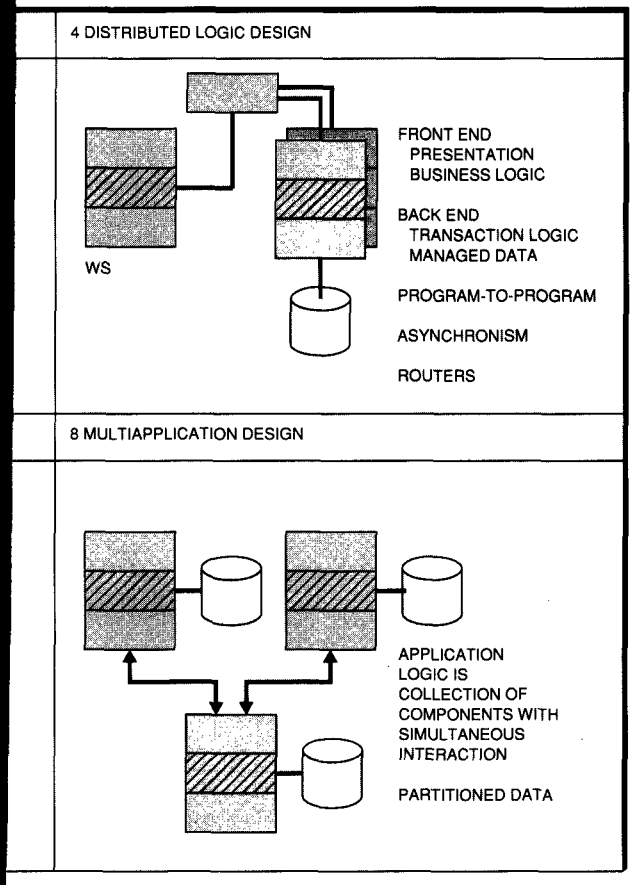
The classification system presented here should not be considered the only correct way to classify distributed systems. Rather, it is a useful device to help us understand why real-world distributed systems assume the structures they do. By taking the perspective of the application program writer, it helps us to understand the system services required to build these distributed systems. It should be further noted that a typical enterprise is likely to utilize more than just one kind of reference design. Mixtures were often observed.

Reference design descriptions

Each of the following subsections describes one of eight types of reference designs.

Nondistributed design. The nondistributed reference design in Figure 3 is included for contrast with the distributed reference designs that follow.

In this design, the physical network consists of a single node with an application program, system



services, and data. The node may be a mainframe, midrange, or workstation computer. The user interacts with this node via an integrated display or an attached display terminal. System services include presentation services for the display and access services for data.

This traditional stand-alone model has generated trillions of dollars in computer industry revenue over the past 30 years.

Remote presentation design. The remote presentation reference design in Figure 4 distributes the presentation service to a remote workstation. The application drives the remote display via an application programming interface. System services produce the protocol and data streams to transport the application commands to the presentation service in the workstation. No application code or "stored data" is in the workstation. The presentation service may be a distributed user interface management system such as the X Window System**, a terminal emulator program, or a distributed dialog manager that renders host dialog screens in a graphical user interface (GUI) format on a programmable workstation.

In this design, the physical network is a workstation connected to one or more back-end systems. The connection may be across a wide area or local area network (WAN or LAN). If the presentation service is a distributed user interface management system, the connection is likely to be limited to a LAN for performance reasons.

The application program runs entirely on the back-end system. The programming interface to the presentation services is invariant, whether the display is local or remote. For this reason, the application in this reference design has the same view of its system environment as the application in the nondistributed reference design.

The system services identified to support this reference design are listed in Figure 4. The nondistributed services required on the back-end systems are local to those systems only. Traditional services such as dialog management, data access, transaction processing, security, and application management are typical. Distributed services require coordination between system software on two or more nodes. The X Window System is an

Figure 3 Nondistributed

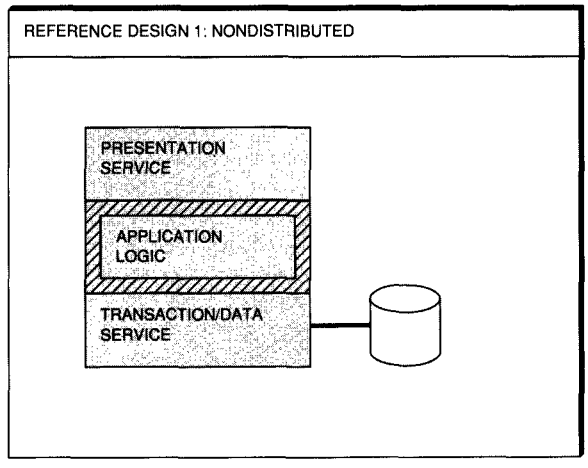
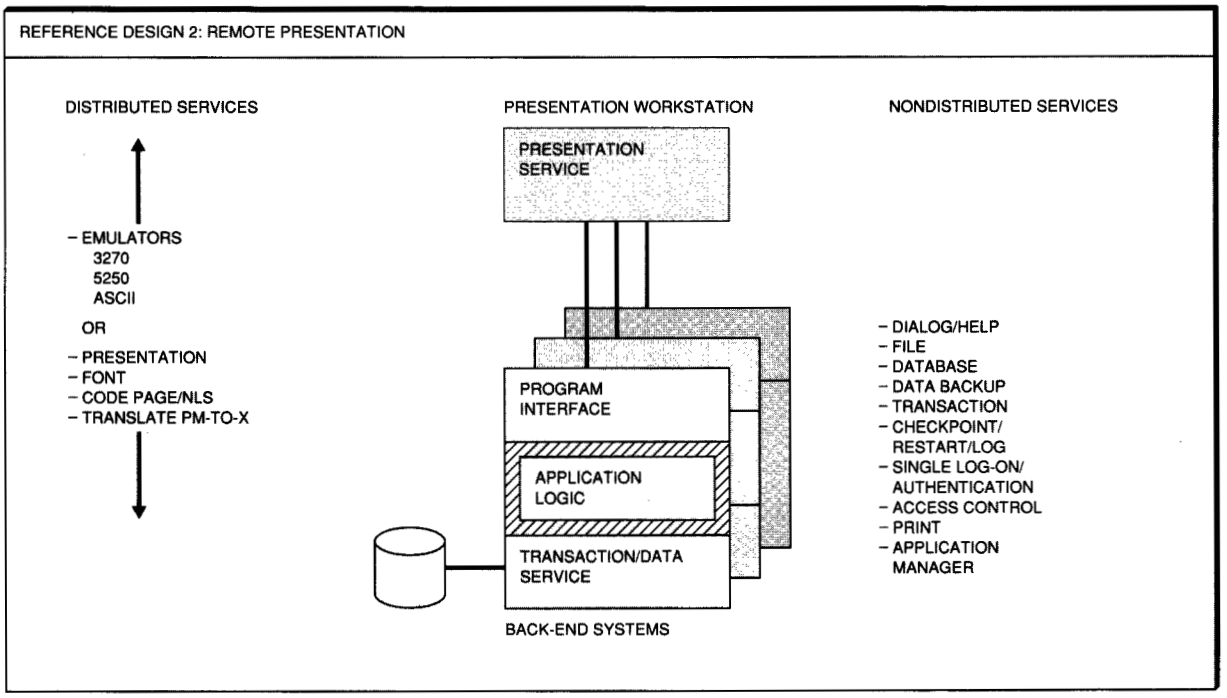


Figure 4 Remote presentation



example of a distributed presentation service. Distributed fonts and code pages are the capabilities to download font generators or code page tables as required.

Remote presentation is often used as the easiest way to attach a workstation to a host. With terminal emulators, the host applications need not be changed. Also, terminal-emulating workstations and display terminals can coexist without requiring two different versions of application software on the host.

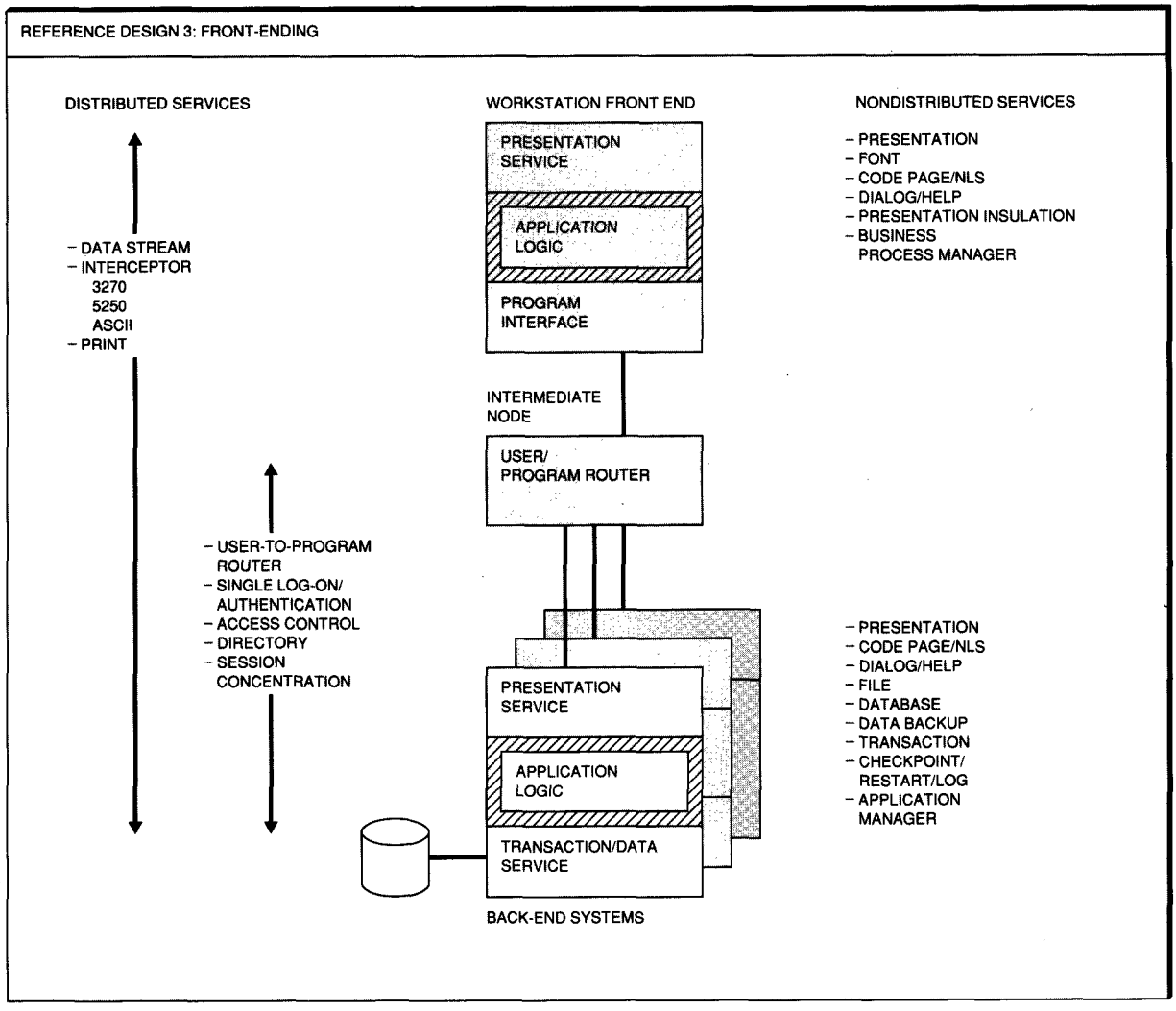
In our survey, a retail enterprise and an insurance enterprise improved headquarters staff productivity by providing workstations on every desk. Users ran personal productivity applications on the workstations (nondistributed) and used terminal emulation (remote presentation) to access the office applications on the host. A graphical user interface adds cut and paste functions to the terminal emulator, giving existing host applications additional functions that the end users never had before. This alone was a productivity aid in some cases.

Remote presentation services like the X Window System are attractive because they support interactions with several back-end systems concurrently. Each session is displayed in a separate "window" on the screen. The X Window System also supports a graphical user interface style, which can improve operator productivity.

A desktop publishing vendor selected the remote presentation design, using a large Advanced Interactive Executive/6000* (AIX/6000*) as the back-end system. The document-editing application required a graphical user interface. By using the X Window System on less expensive AIX/6000 models and X-terminals, the vendor was able to have the cost of the back-end system shared among a work group of users. Also, because of the popularity of the X Window System, the editing application was accessible from a variety of vendors' workstations.

Front-ending design. The front-ending design in Figure 5 is different from the remote presentation design because the workstation has "front-end" application logic. This design is often used to

Figure 5 Front-ending



layer new application function in front of existing back-end applications. In this way, value can be added to extend the existing application inventory.

The front-end application intercepts and interprets the display data stream generated by the back-end systems. The front-end application logic then re-presents information on the workstation display. The user interface is written twice: once on the back end, and once again on the workstation.

The front-end application logic may be used to simply transform the user interfaces of each back-end system into consistent and more modern graphical user interfaces. The options available on the back-end systems may be presented as a menu, thus producing an easier-to-use, single system image to the user.

For example, a user might correct an error in a customer billing by following these steps:

1. Verify bill was issued in the amount claimed.
2. Apply credit to customer's account.
3. Confirm correction and print acknowledgment for mailing.

Back-end transaction programs to query the billing data, update the customer account data, and log the corrections would be selected from the user's menu. This sequenced transaction scenario consists of short interactions with transaction programs on (possibly different) back-end systems. Customers have evolved several services to support this model.

The first service required is a router function that connects the workstation to each back-end system as needed. The router uses a directory to locate the desired system and then routes the LU 2 (3270), LU 7 (5250), or ASCII (TTY) protocols and data streams accordingly.

Response time considerations may require the router service to maintain continuous sessions with the back-end systems, even though the sessions may be used only intermittently. In this case, an intermediate router node can be used to provide session concentration. The router maintains one session with each back-end system and temporarily allocates that session to a workstation wishing to make a request. If the router function is placed in an intermediate concentrator node, session concentration reduces the overall number of standing sessions required.

Single log-on eliminates the need for the user to provide a separate password to each back-end system. The user logs on once to a single log-on service. An access control service checks to see if the user is authorized to access the desired back-end system. An authorized user can be authenticated to the back-end system, thus avoiding another log-on or password interaction with that system. The router, directory, single log-on, access control, and authentication services may be supported on the workstation node or (optionally) on an intermediate node. These services can be designed on an intermediate node in such a way as to support limited function display terminals as well as workstations.

In the customer billing example above, the user selected each step in the process. More sophisticated front-end applications can actually automate this process. A business process management service provides support by which the application is constructed as a sequence of *scripted* business steps. Recovery support may be automatically provided if one of the steps in a sequence fails to complete.

Finally, as the usefulness of the front-end application grows, software developers seek to "port" these applications to many different workstation platforms. Some have devised presentation service insulation layers that isolate the application logic from variations in the presentation services provided on different workstations.

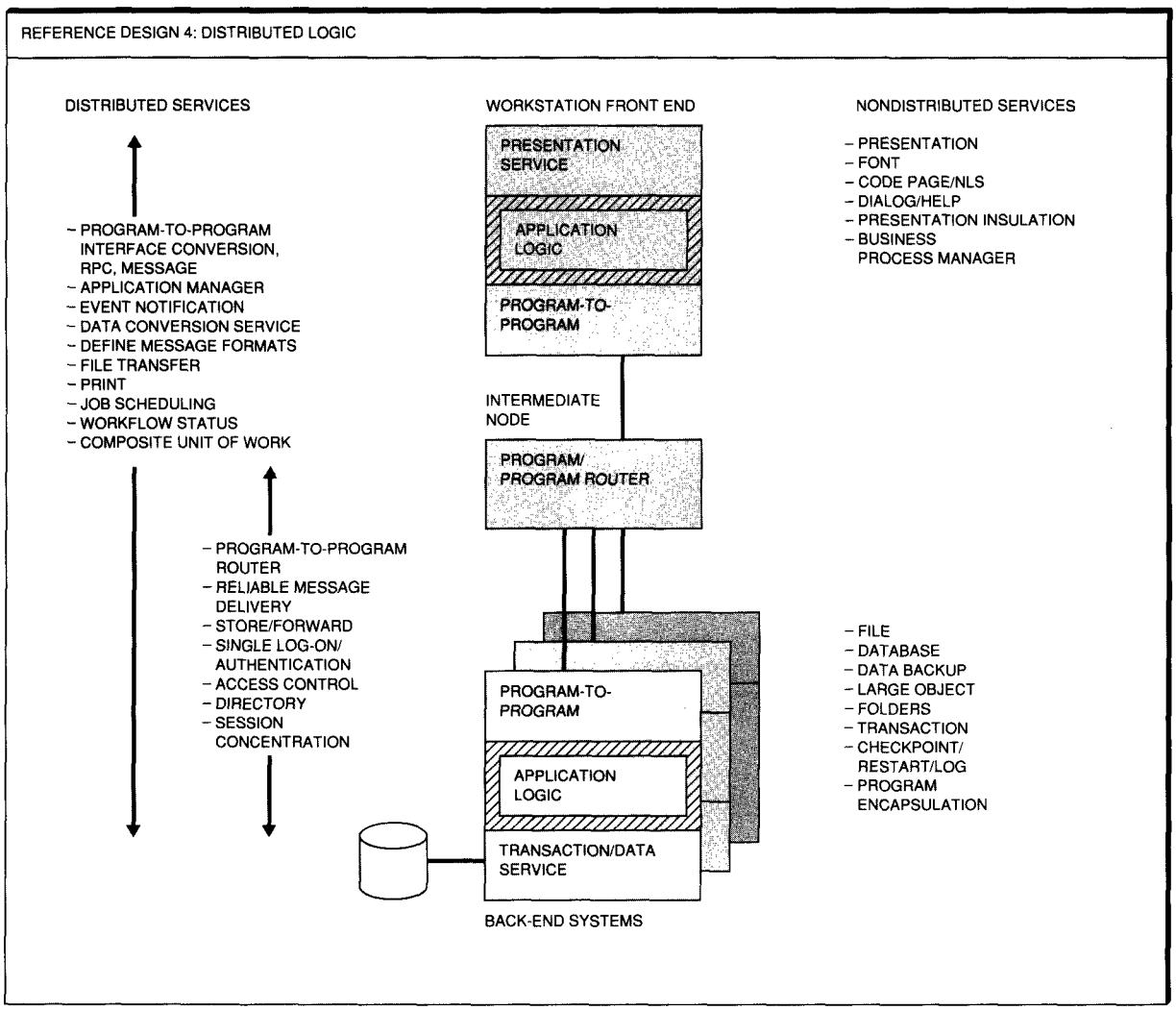
Customers use the front-ending reference design to provide users easier access to many different existing back-end systems. A large hospital used Easel for OS/2** to provide a consistent, windowed user interface to Information Management System (IMS*), DATABASE 2* (DB2*), and Digital Equipment Corporation VMS** back-end systems.

A telephone service center is replacing display terminals connected to multiple IMS back-end systems with AIX/6000 workstations. The front-end application will automate 36 different operator activities. Depending on the caller's complaint, the operator selects an activity and follows a dialog through to completion. All interactions with the back-end transaction programs are handled automatically. The telephone service center expects this front-end application logic to reduce training time for these high-turnover positions, as well as to improve customer service and reduce transcription errors.

A large insurance enterprise needed to provide access for thousands of users to hundreds of different back-end systems. Agents and claims adjusters in the field would use reduced-function terminals to dial into a central processor, whereas office staff would use more capable workstations. An intermediate node provided the router, session concentration, directory, single log-on, access control, and authentication services so both types of users could be supported. The resulting concentration of directory and access control files on the intermediate node simplified administration. The users enjoyed menu-driven, single log-on access to all the back-end systems for which they were authorized.

Distributed logic design. The distributed logic reference design in Figure 6 is similar to the front-ending reference design. The difference is that the front-end and back-end application components interact with each other directly via a program-to-program communication service. The style of

Figure 6 Distributed logic



this service varies, from conversations, to remote procedure calls (RPC), to queued messages.

In the front-ending design, the back-end application is usually an existing application designed to drive a display terminal. In the distributed logic design, the back-end application does not have any user interface component, and so it cannot function as a stand-alone application. It is usually a new application, using a programming interface supported by the program-to-program communication service.

An important constraint is that the stored data are still centralized on the back-end system. This distinguishes the distributed logic design from other designs to follow. The business needs of the enterprise may dictate this constraint for several reasons. Enterprises like finance, insurance, and retail companies may be organized in a hierarchical fashion, with a central database and branches. Such data are usually considered mission-critical, where integrity and security are essential. Well-established recovery services on the back-end systems like archiving, checkpoint image copy,

restore utilities, and alternative-site disaster recovery cannot be easily applied if the database is distributed and the branches lack computer operations staffing. Also, branch offices usually cannot provide the degree of physical security required for these data. And finally, the database may be large and not easily partitioned because of the transaction reference patterns.

An important constraint is that the stored data are still centralized on the back-end system.

Given these constraints, the distributed logic design seeks to strike an appropriate balance between the front-end and back-end applications. The program-to-program interface allows more flexibility in selecting this balance point than the front-ending design. For example, all presentation services, including dialog management and help services, can be off-loaded to the workstation. On the other hand, communications bandwidth and latency force any application logic requiring high-bandwidth access to the data to remain on the back-end system. Performing a high-bandwidth transaction such as sorting data across a network should be avoided in favor of performing the sort local to where the data reside and shipping only the results across the network. Typically, the back-end systems become transaction servers.

The message style of program-to-program interface introduces another degree of flexibility. Unlike the synchronous connection-oriented relationship in the front-ending design, the queued messaging model supports connectionless asynchronism. That is, the front-end application can send a message to a back-end application and continue to do additional work while waiting for a reply. In fact, it may not even expect a reply. Because messaging is connectionless, the server does not need to be running at the same time as the application (i.e., there is no need for an established connection at run time). This style of designing asynchronous, cooperating applications generally requires an event service to notify a designated application of both expected and unexpected events and exceptions. Also, the front-end application is no longer limited to sending its requests to back-end applications one at a time. Several request messages may be sent to different back-end systems concurrently, enhancing user-response time over a synchronous mode of execution.

Any business process management service in the workstation must consequently support scripts with asynchronous and possibly concurrent steps. In addition to event notification, services such as data or time-dependent job scheduling and workflow status monitoring may be used. There may be sets of concurrent steps that must all participate in the same unit of work. These sets are referred to as a "composite unit of work." An automatic recovery service is needed to perform the necessary rollback if one of a set of concurrent steps in a composite unit of work should fail to complete.

Messages and remote procedure calls emphasize even more the short interactive nature of the request from front-end to back-end applications. The role of the intermediate node takes on more importance in this reference design. Compared to the user-to-program router in the front-ending reference design, the router service to support distributed logic must route an increasing number of short program-to-program requests. It may be required to accomplish this routing over existing conversation networks like Systems Network Architecture (SNA) with either a limited number of available sessions or high overhead for establishing and relinquishing a session, thus making session concentration essential. Also, customers may need support for storing messages when the recipient system is not available. A reliable message delivery service guarantees that a routed message will never be lost in the event of delivery service failure. This guarantee is usually accomplished via transaction processing techniques such as cascaded units of work.

The use of program-to-program interfaces introduces the need for a set of associated services. An incoming request to start a conversation, run a remote procedure, or receive a message means that an application must be dispatched to handle

the necessary interaction. The application manager does this job and also terminates the application when the request is completed. Data conversion services alleviate the application's burden of translating the data passed from a dissimilar system. Methods to define data formats are necessary for automatic data conversion and useful for application-to-application consistency. Services to support encapsulation would enable old back-end applications to interact with new front-end applications via the program-to-program interfaces.

The distributed logic designs selected by customers and vendors in our survey started with the assumption that the data will remain on the back-end system. An application vendor to the insurance industry established a framework of services to support a distributed logic design. Workstations replaced display terminals in the offices of agents. Message services connect the workstations to back-end Customer Information Control Systems (CICSs*) or DB2 systems, where the databases are maintained. The back-end applications were rewritten as message-driven transaction programs.

A manufacturing enterprise is off-loading engineering design applications to Operating System/2* (OS/2*) workstations. The bill-of-material data will remain in the back-end IMS systems, where it can be shared and secured. The front-end application interacts with the designer to query and update the bill of material. The connection from front end to back end in this case is LU 6.2, a conversation model. Prototype implementations have been instrumented to ascertain splits between the front-end and back-end application logic so as to minimize communication bandwidth and maximize performance.

Because of the shared nature of the information between engineers, plant floor operators, and managers, this customer also required that AIX users have access to the same front-end applications. Porting the applications from the OS/2 (Presentation Manager*, or PM) environment to the AIX (X Windows System) environment was avoided in this case by providing a specialized PM-to-X protocol converter on the OS/2 workstation, thus supporting indirect access from AIX users.

For both the manufacturer and the insurance application vendor, the data on the back-end systems will consist of collections of different kinds of data, including documents, structured data, and image data. The insurance applications will use a folder model to collect all of the policy information for a given customer, whereas the manufacturer's part designs include application design files, graphics, and such structured information as a bill of materials. In both cases the back-end data storage systems are required to support these collections of dissimilar data.

A reliable message delivery service guarantees that a routed message will never be lost in the event of delivery service failure.

A national retail enterprise connected a CICS OS/2 workstation in every store to a back-end Customer Information Control System/Multiple Virtual Storage (CICS/MVS) system. The customer records are consolidated and shared in a back-end system so that a customer need not return to the same store of purchase for service. The front-end application supports an easy-to-use dialog interface with the clerk and automates the steps involved in a given customer service transaction. A CICS program-to-program interface is used by the front-end application to invoke transaction programs on the back end. This interface was chosen for its economical use of communications bandwidth.

The telephone service center application that used the front-ending reference design will eventually be replaced by a distributed logic reference design. The requirement is for new back-end transactions to use a program-to-program message interface rather than the IMS presentation services for display terminals. All presentation services support will be off-loaded to the front-end AIX workstations. Furthermore, the customer requires a message router for session concentration and to add flexibility in routing messages to alternate back-end systems.

Data staging design. The front-ending and distributed logic reference designs may have drawbacks if the number of workstations puts a heavy load on the back-end databases or if the distance between the workstation and back end increases the communication latency and cost. Communication breakdowns can put the workstation out of business until communications are re-established.

The data staging reference design in Figure 7 solves these problems by moving the required data closer to the front-end workstations. With data staging, the "true" data remain on a regional back-end system while snapshots of portions of the data are staged (i.e., downloaded) to local back-end systems. The regional and local back-end systems are often called regional and local servers. Data staging solves the problems noted above but still preserves the security and integrity of the "true" database by keeping it under the protection of a central back-end system where it can be backed up and consistently recovered.

Data staging maintains the availability of some operational data even if the regional server is down. It can improve access time to the local server, reduce communications cost to the regional server, and reduce load on the regional server. In contrast, it involves additional design complexity to devise the staging policies and additional operations complexity to periodically stage the data snapshots to the local servers.

Various data staging policies have been devised, depending on the characteristics and expected uses of the business data. The simplest policy in our customer survey is to periodically stage read-only data to the local servers. In this one-way flow of information, the frequency of refresh depends on the requirement for data currency of the front-end application. Securities trading applications require frequent refreshes of rates and price quotes. This policy is referred to as a "data feed" and may be implemented using periodic network broadcasts of data. Interest rate tables used by finance applications may only require daily refresh.

Another policy is to identify a subset of a database to be staged to each local server. Application reads are satisfied by the local server if possible or routed to the regional server if necessary. Application updates are always "write-through" to both the local and regional servers. This policy is referred to as "contingent staging" and may be used to stage data that tend to be affiliated with each branch office, with a low requirement for currency. A nightly refresh rate is typical for this type of data. The write-through of any update maintains the moment-to-moment accuracy of the regional database. Obviously, care must be taken when the update of another branch invalidates the staged data!

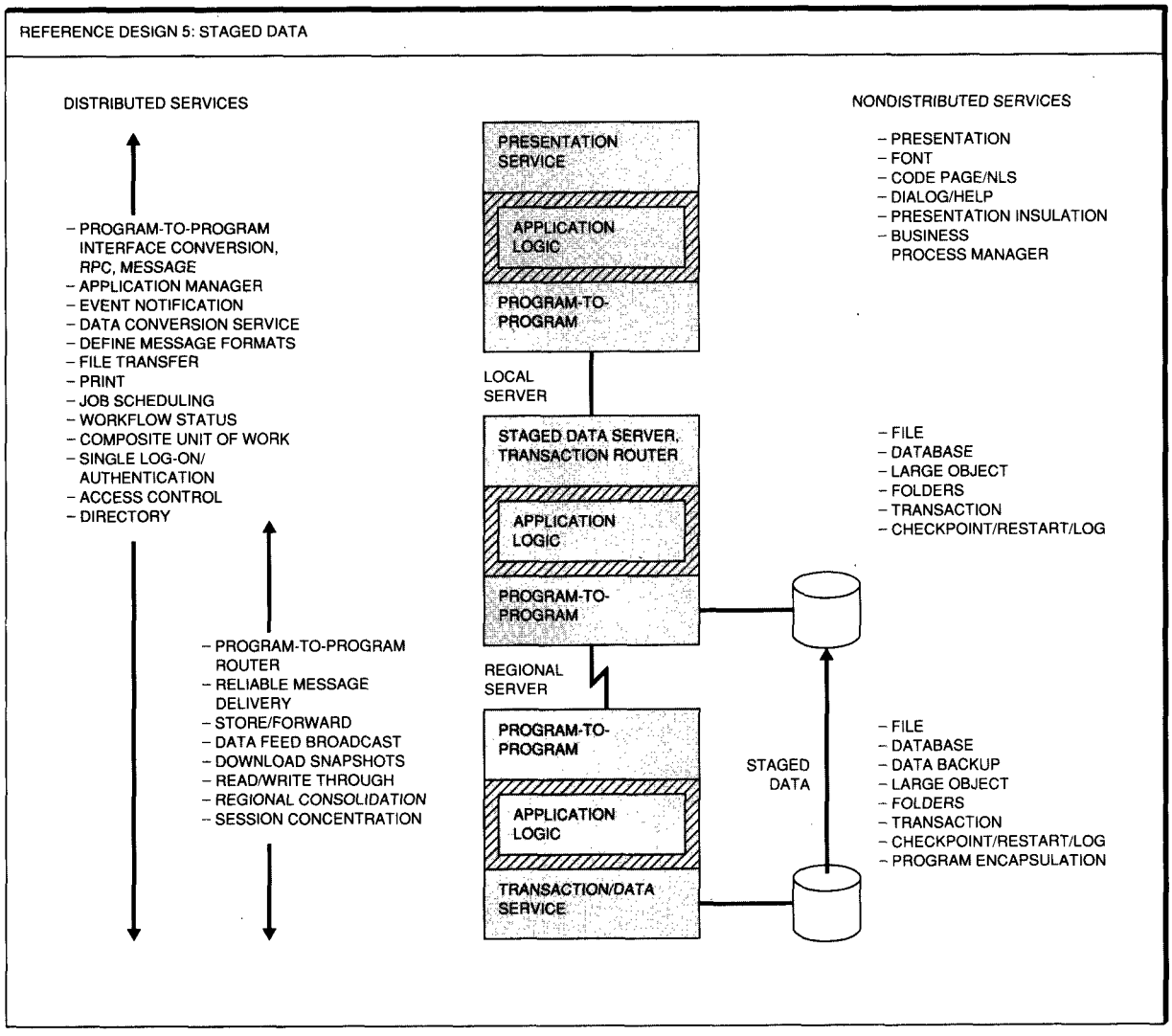
A "consolidation policy" relaxes the condition that the regional database be accurate from moment to moment. With this policy, the data are staged to each local server, and reads and updates are satisfied locally. The updated local data are consolidated to the regional database periodically. This policy may be used when the regional database can be partitioned for each branch with no sharing between branches. If sharing does occur, out-of-date data must be tolerated. The regional database might be used for daily or weekly report generation.

Other variations of data staging policies are possible, and more than one policy may be used by a customer. Our survey identified various services customers have devised to implement these policies. A broadcast message service was used to implement the data feed policy. A snapshot download (with version control) is required by the contingent data and consolidation policies. The consolidation policy also requires a regional data update consolidation service.

The local server node in Figure 7 plays a substantially larger role than the intermediate nodes in the previous reference designs. In general, the local server must have the same database characteristics as the regional server. The local server also has application programs to access the staged data. Finally, the program-to-program router service must route requests from the front-end application to the appropriate local or regional server applications. Depending on the staging policies in effect, the request may be routed to the local or remote server or both.

A large bank with many branch offices had implemented a distributed logic design using the IBM Financial Branch System Services (FBSS) product on DOS workstations connected to IMS back ends via an IBM 4702 financial controller. The bank plans to have the current design evolve into a data staging design by adding an OS/2 local database

Figure 7 Data staging

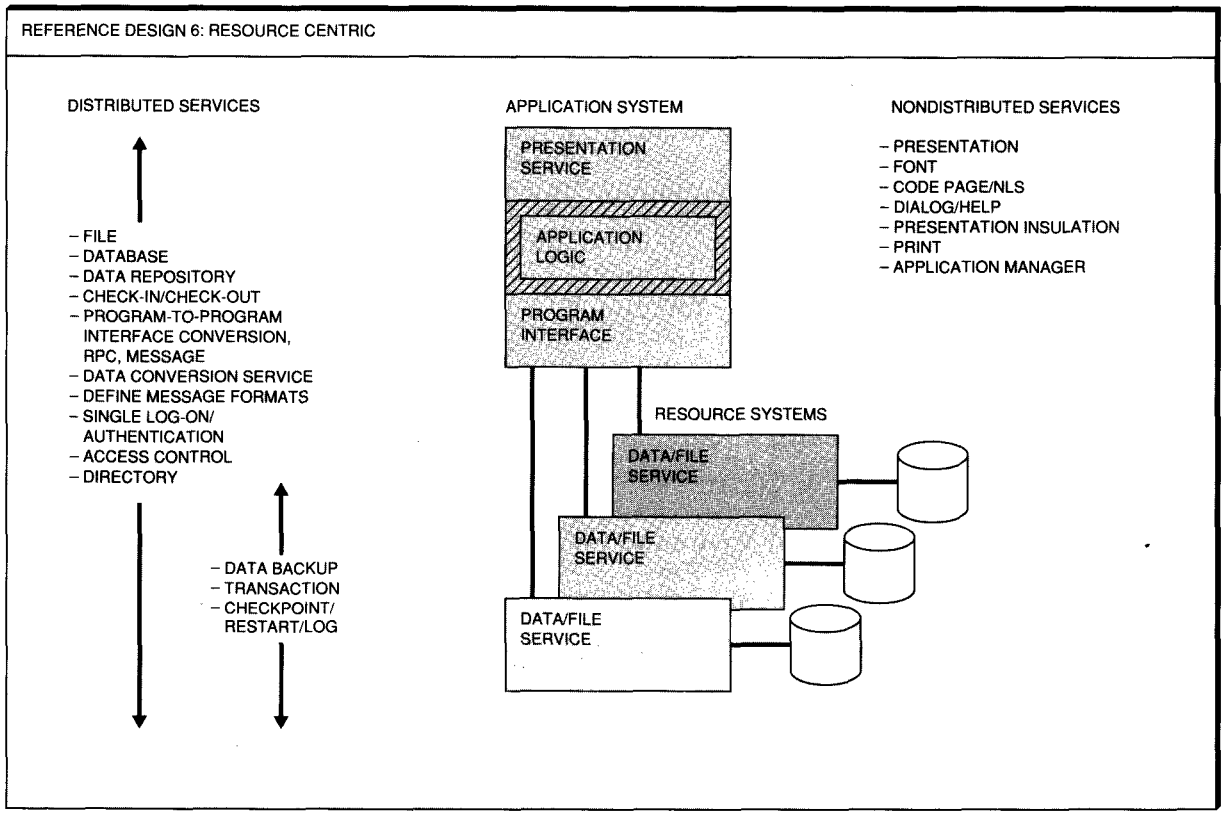


server in each branch. A study has identified a subset of the regional database that will be downloaded to each branch nightly. This contingent data policy is expected to improve customer service availability in the event of failure to access the regional server.

A semiconductor manufacturer plans to use a data staging design to run as much work as possible on local work group servers. An analysis of the shared data characteristics determined that a distributed logic design was required for some data but that contingent staging and consolidation policies could be used for other data. This customer also required the option to support asynchronous write-throughs to the regional server, so the front-end application could continue to work while waiting for the update to complete.

Resource-centric design. The resource-centric reference design in Figure 8 supports direct, remote access from an application to some system resource, usually data. The application accesses the remote resource via the same programming in-

Figure 8 Resource centric



terface used to access the local resource. Examples include the local file system interface for files, a Structured Query Language (SQL) interface for relational databases, or a check-in/check-out interface for data repositories. Different files or relational tables may be located at different remote nodes. Underlying system services do the job of locating, accessing, and returning the resulting data.

This reference design is the only one in which an application has direct access to remote data. In the front-ending, distributed logic, and data staging reference designs, the front-end application talks to a remote application, which in turn accesses the data.

Well-known products such as Novell NetWare** and Sun Network File System** have popularized this reference design. In addition to file servers, database vendors like Oracle and IBM support remote access to database servers. The resource-centric reference design is probably the most commercially successful distributed reference design to date. It has probably enjoyed early success because it is an easy way for separate users to share a resource. Workstation users can easily exchange data through a shared file or database server. Since the application accesses both local and remote data as if all data were local, remote file and database servers can be introduced without modification to existing applications. Users may also share such devices as printers or plotters to lower the cost of the device per user.

A resource-centric reference design works best when the amount of data accessed is easily supported by the communication bandwidth available. A low-latency network environment such as a high-speed local area network is desirable. When communication delays are intolerable, or when additional path lengths due to the commu-

nication protocols become a significant computational consideration, it is often more efficient to place part of the application logic local to the data. As hard as it may seem to believe that a LAN may not have enough bandwidth, we observed one insurance company that wrote a LAN system in which every transaction created thousands of file I/O requests. They could not understand why they where getting a response time of two minutes per transaction. In short, the message traffic created by their application was too much for redirected file I/O activity to handle. In these cases, the previous reference designs are preferable.

The process-driven reference design is the first where both application logic and data are permanently split apart and distributed in paired pieces.

In the resource-centric reference design, system services such as file, database, repository, and print must be extended to support remote access. If the data are spread across multiple nodes, additional care must be taken by the system to ensure integrity and recoverability. Underlying services such as unit-of-work, backup, checkpoint or restart, and logging must now work in a coordinated fashion against the distributed pieces of data.

Popular file, database, and print servers are now widely used and were observed in our survey. One manufacturing customer also created a repository for engineering design files. Workstations can check in or check out design files from the remote repository via a programming interface on the workstation. Once checked out, the design files are transferred to a file server, where they are shared by the work group.

Process-driven design. The process-driven reference design is the first design where both application logic and data are permanently split apart and distributed in paired pieces. Each node in Figure 9 supports both application code and its own stored data. The data are private to the application and are not directly accessible by applications on other nodes. Each application-data combination accomplishes a discrete task.

The process-driven reference design is characterized by a stepwise execution of the applications. Each application may be viewed as a job step in a higher-level process, which is designed to accomplish a business-oriented task.

The simplest process is a set of sequential steps. Each application in the sequence must complete before the next begins. More complex processes define parallel sequences of steps that may execute concurrently.

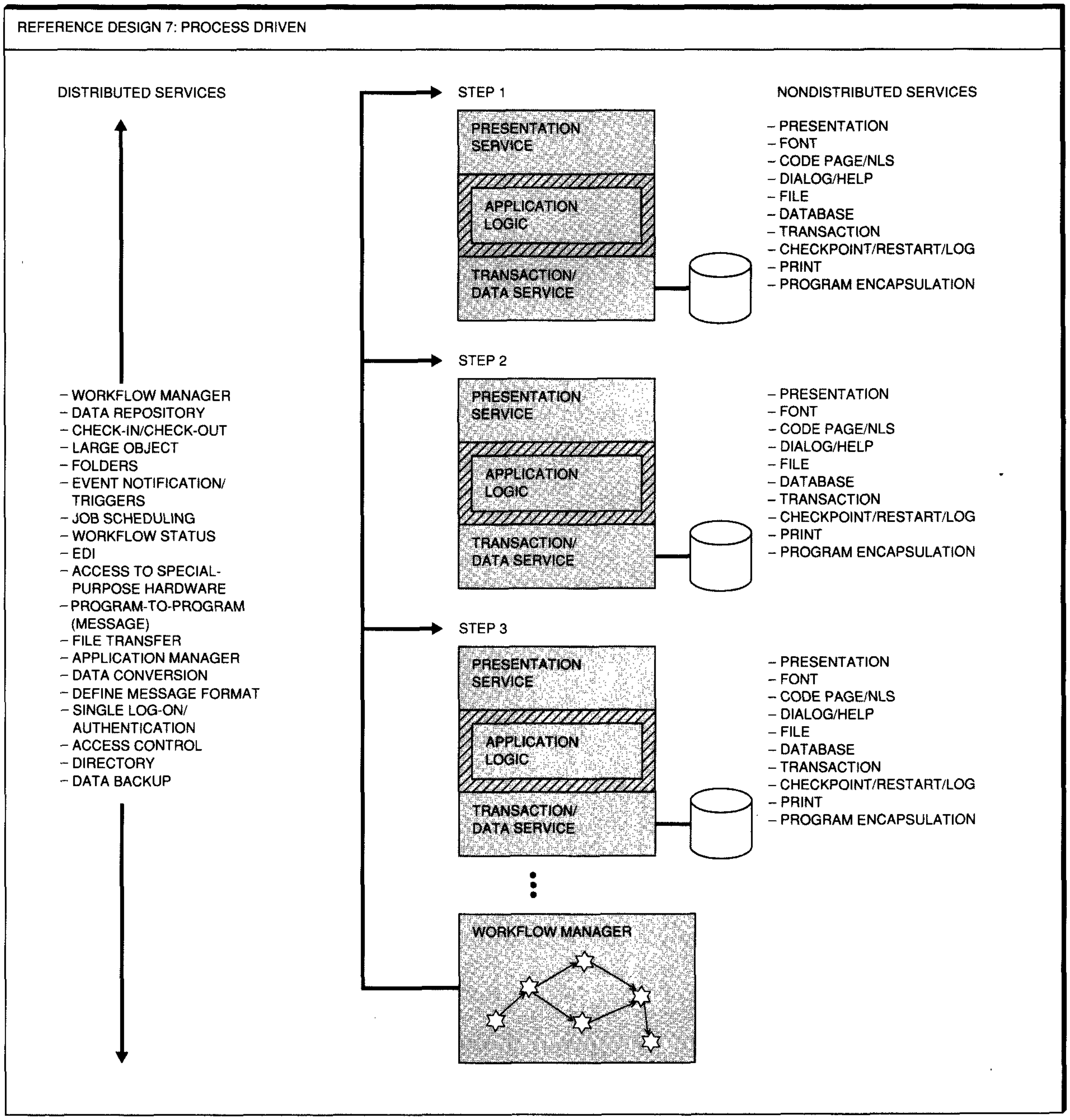
The applications do not engage in simultaneous communication with each other. Exchanges of information between applications usually occur via a prepared data handoff. The handoff may be implemented with I/O files, or locking and trigger conventions in a shared data repository.

A simple process may be executed by manual convention. An example of this is the process followed to prepare large-scale scientific simulation jobs. The first step is to prepare the model description on a workstation. The second step is to submit the simulation run to a high-performance supercomputer. The final step is to review the results of the simulation using visualization applications on the workstation.

Another example from the banking industry is check processing. A check drawn on bank A and submitted for deposit to bank B will be consolidated into a debit summary for bank A. Accounts are debited in bank A, and funds are transferred to bank B, where the appropriate accounts are credited. The steps in this process are followed by established business convention. Some system services such as job control languages are useful in this highly repetitious scenario.

Many customers are carefully defining their business-oriented processes to capture them for automatic execution. Computer-integrated manufacturing is an example of where the processes are more complicated than the previous examples. In this case, additional services are required to support the process-driven reference design. The workflow manager is a facility that regulates

Figure 9 Process driven



the overall execution of the business process. The process may be defined to the workflow manager by an explicit script language or a defined set of events and triggers. The workflow manager implementation may be centralized or distributed.

In addition to the job control language, file transfer, and shared repository services mentioned above, the workflow manager may also require application manager, event notification, job scheduling, and workflow status services. The workflow man-

ager function may be implemented as part of a data repository design. In this case, events are defined as changes to data values. Data triggers invoke application code or send messages to execute the next step in the process.

The workflow manager may appear to be similar to the business process manager defined in the front-ending reference design. We have chosen to use these different terms to allow for the differences in emphasis we have observed. The front-end business process manager is located on a front-end workstation and serves as an agent for one user. It translates high-level user commands into multiple back-end transaction requests. The workflow manager is an autonomous monitor of defined workflow processes. Its scope may extend to the activities of hundreds of users. It will initiate action based on events other than human commands. It is often implemented on a host system and sometimes integrated with a data repository.

In the process-driven reference design, a message style service often seems to be preferred over conversations or remote procedure calls. Messages are used to transmit events, status, job step commands, and hand off data. Many business processes such as ordering and billing involve interactions with entities outside of the enterprise. Automating these processes requires support for message standards such as EDI (electronic document interchange). The automated process may require coordination with special-purpose hardware such as a robot controller or an ATM (automatic teller machine). Messages and other services must be supported on these platforms also.

Multiapplication design. The multiapplication reference design in Figure 10 also has application logic and stored data split apart and distributed in paired pieces. The data are private to the application and are not directly accessible by applications on other nodes.

The multiapplication reference design is different from the process-driven reference design because the applications are active at the same time and engage in simultaneous communications with one another. Each application works as part of a team of applications to finish a given business task. Since the stored data of each application are not directly accessible by other applications, this reference design is suggestive of an object-oriented design style in which each application *encapsulates* its data. It is often referred to as the "most distributed" reference design or "most general" reference design. It would appear to offer the most freedom in distributing application logic and stored data. In fact, it seemed to be used only in environments where the constraints left no other choices!

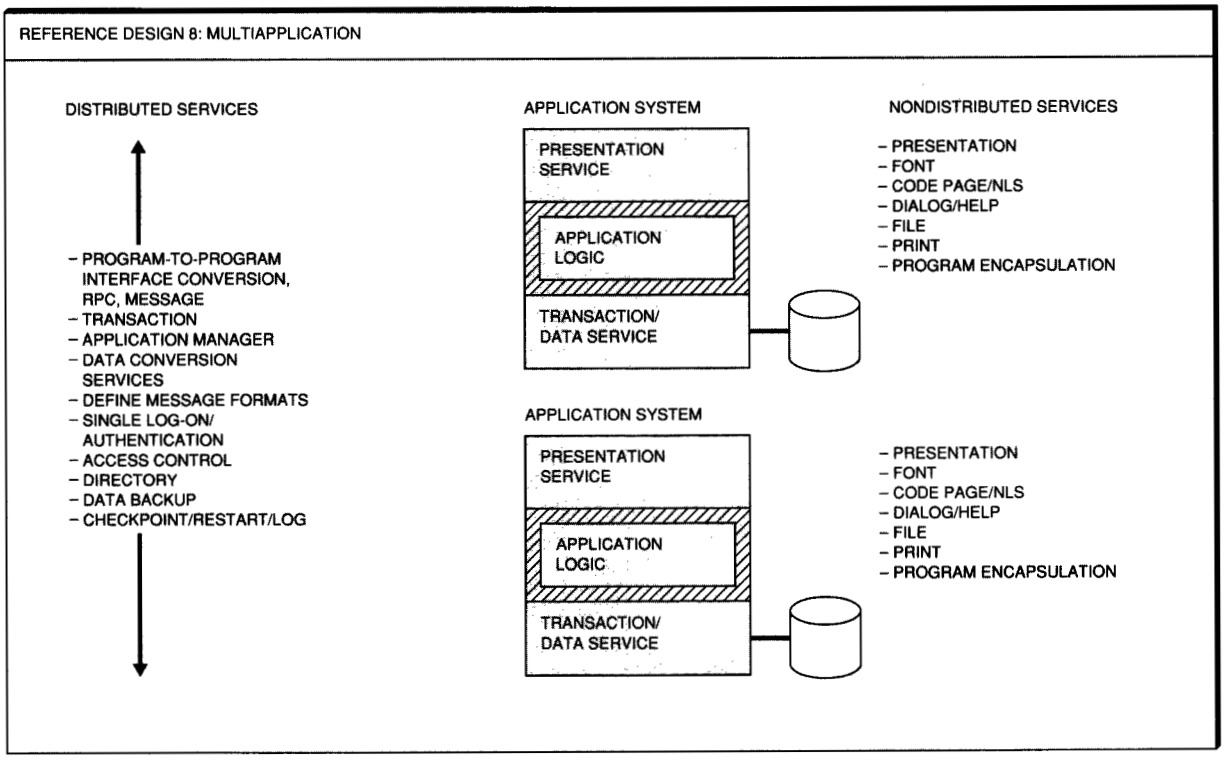
For example, a large hospital had different vendor systems in place for each department. The hospital wanted to tie these systems together so that information would flow easily from the application of one department to another. The goal is to make all information about a patient easily accessible to doctors, nurses, and administrators. Transcription errors from copying data by hand will be eliminated, and patients will receive better quality care.

In this environment, the hospital requires techniques to encapsulate the old applications with new software so they can respond to requests from other programs as well as from terminal operators. Because of the existing framework, there is little opportunity to make the kind of distribution trade-offs discussed in previous reference designs.

Customers in transaction environments occasionally find that they must run applications against databases that are separated but must remain coordinated. The databases may be in different departments or even different enterprises. They may have been implemented separately before there was a need for accessing both together. The data access bandwidth requirements may be such that a resource-centric reference design would perform poorly. The only alternative is a multiapplication design, where applications in each node access their databases and interact with one another to update them in a coordinated fashion.

Coordination requirements make these applications more complicated to design. Data integrity requires that if one application fails to update its database, the other application's updates to its database must be undone to restore consistency between the two databases. Such designs require a distributed transaction management service to maintain database coordination. The technology is called distributed unit of work and is usually implemented with two-phase commit protocols. Furthermore, larger-scale recovery procedures such as backup, logging, and checkpoint or re-

Figure 10 Multiapplication



start must be distributed to support recovering each database to a point where it is coordinated with the others.

System management and application development

The requirements described in Figures 4 through 10 and summarized in Table 1 concentrate only on the services necessary to support an operational reference design.

Application development tools and system management support are equally important requirements for the successful deployment of a reference design. For each reference design there must be tools to design, code, test, and debug the various application components targeted for each hardware platform. Efficient management of the increased complexity of these multiplatform solutions is also needed. Although these requirements are beyond the scope of this paper, we can make the following observations.

Application development. Since the distribution in the remote presentation and resource-centric designs is transparent to the application, tools for developing stand-alone applications should work equally well for these designs. Other reference designs, however, present new challenges. The distributed application components in the remaining reference designs must be developed in concert to interoperate properly. It is probably unacceptable to require each application component to be developed in isolation of the others. It is even worse if different tool environments must be used for each application component, depending on which system is targeted! Tools for design, code, test, and debug should support a cohesive view of the application components yet permit multiple back-end code generation for each hardware platform.

The front-ending design requires new front-end applications to intercept the display terminal data streams produced by old applications. When old design information is no longer available, it pres-

Table 1 Summary of system services required by reference designs (grouped by functional category)

System Services	Remote Presentation	Front-Ending	Distributed Logic	Staged Data	Resource Centric	Process Driven	Multi-application
Presentation services							
Emulators (3270/5250/ASCII)	D						
Display presentation	D	X	X	X	X	X	X
Selectable fonts	D	X	X	X	X	X	X
Code page/national language	D	X	X	X	X	X	X
Dialog/help functions	X	X	X	X	X	X	X
Presentation insulation		X	X	X	X		
Translate PM to X (OS/2)	D						
Print	X	D	D	D	X	X	X
Data stream interceptor		D					
Data services							
File system	X	X	X	X	D	X	X
Database	X	X	X	X	D	X	X
Data repository					D	D	
Check-in/check-out					D	D	
Large object			X	X		D	
Folders			X	X		D	
Automatic backup	X	X	X		D	D	D
Transaction processing							
Transaction manager	X	X	X	X	D	X	D
Checkpoint/restart/logging	X	X	X	X	D	X	D
Composite unit of work			D	D			
Application services							
Program-to-program interface							
Conversation			D	D	D		D
Remote procedure call			D	D	D		D
Message queues			D	D	D	D	D
Data feed broadcast				D			
File transfer			D	D		D	
Electronic document interchange						D	
Data conversion services			D	D	D	D	D
Define/register message formats			D	D	D	D	D
Application manager	X	X	D	D	X	D	D
Event notification (alert/exception)			D	D			
Program encapsulation			X	X		X	X
Access to special-purpose hardware						D	
Security							
Single log-on/authentication	X	D	D	D	D	D	D
Access control	X	D	D	D	D	D	D
Routing services							
User-to-program router		D					
Program-to-program router			D	D			
Directory		D	D	D	D	D	D
Reliable message delivery			D	D			
Store/forward			D	D			
Session concentration		D	D	D			
Process management							
Business process manager		X	X	X			
Workflow manager					D		
Programmed job scheduling			D	D	D		
Workflow query/status			D	D	D		
Event notification (status/triggers)						D	
Data staging services							
Download snapshots				D			
Regional consolidation				D			
Read/write through				D			

X = Function required is nondistributed; D = Function required is distributed

ents a difficult job of re-engineering. A tool to analyze the data streams is a productivity requirement.

The distributed logic and data staging designs involve a delicate balance between front-end and back-end function placement. Performance prediction and analysis tools are required. The routers in these designs will require application-level interfaces to support customer-supplied routing algorithms. Network-level traffic analysis will help define these algorithms. Also, database analysis tools will be required to evaluate what subsets of data may be staged to local servers.

The process-driven and multiapplication designs require tools to formalize and catalog the data and message interchanges between the diverse applications. The multiapplication design in particular needs tools to encapsulate old applications so that they fit into the defined request interchange framework.

System management. System management challenges for reference designs range from mundane operations to sophisticated new approaches. Customers have experienced difficulty even with remote presentation designs, where the only connection to the back-end system is a terminal emulator. Basic operations like software distribution, configuration management, problem reporting, and data backup have proved to be considerable challenges when thousands of workstations are involved.

More challenging requirements lie ahead. Cost pressures to reduce operational staff will lead to focal point concentrations of management support. From these focal points, operators must be able to manage distributed reference designs as they manage stand-alone systems today. Interoperable software components place increased demands on coordinating software upgrades. Backouts of other software component upgrades may be necessary if the upgrade of one component fails. When a user request can involve transparent access to several distributed systems, new system-wide approaches to user enrollment, authentication, and access control must be supported. Customers have asked for new accounting metrics based on transaction identifiers, so back-end server usage can be billed accordingly. Hardware and software problem reporting to a remote focal point must be accompanied by the ability to diagnose and repair remotely. And finally, data management capabilities must be extended to data in remote databases. Separate databases in the resource-centric, process-driven, and multiapplication designs may still require backup, archive, checkpoint, and restart operations to be executed in a coordinated fashion to retain consistency.

Conclusions

Summaries of all the operational system services required by each reference design are shown grouped by functional category in Table 1. Table 2 shows just the distributed services ordered by their frequency of occurrence in the seven reference designs. Tables 1 and 2 illustrate that the reference designs share common requirements to a high degree. Fifty-six percent of the required services are used by more than one reference design.

In general, every system service required by more than one reference design should be designed to be a common service, which can meet the particular needs of each reference design. With such an approach, the items listed in Table 2 can be viewed as the building-block elements of a distributed services architecture that can be used to construct any reference design.

The stated objective of this paper was to define a set of application reference models that could be used as a starting point (i.e., template) for designing distributed applications. The development of the reference designs has led us to the following propositions:

1. The reference designs contain elements of client/server and cooperative processing design philosophies, but they suggest that a more general term is needed to encompass the solutions depicted. We suggest the term distributed systems.
2. The reference design classifications constitute a reasonably complete description of the kinds of real-world solutions customers require—that is, solutions that are distributed, yet provide cross-enterprise integration. Customers may use these reference designs as guides to designing distributed system solutions tailored to their own particular needs.
3. A common architecture of "building block" distributed services is the means by which we

Table 2 Summary of distributed system services required by reference designs (ordered by occurrence)

Distributed System Services	Remote Presentation	Front-Ending	Multi-application	Process Driven	Resource Centric	Distributed Logic	Staged Data
Single log-on/authentication		D	D	D	D	D	D
Access control		D	D	D	D	D	D
Directory		D	D	D	D	D	D
Message queues			D	D	D	D	D
Data conversion services			D	D	D	D	D
Define/register message formats			D	D	D	D	D
Conversation			D		D	D	D
Remote procedure call			D		D	D	D
Application manager			D	D		D	D
Print		D				D	D
Session concentration		D				D	D
File transfer				D		D	D
Programmed job scheduling					D	D	D
Workflow query/status					D	D	D
Automatic backup			D	D	D		
Composite unit of work						D	D
Event notification (alert/exception)						D	D
Program-to-program router						D	D
Reliable message delivery						D	D
Store/forward						D	D
Data repository				D	D		
Check-in/check-out				D	D		
Transaction manager			D		D		
Checkpoint/restart/logging			D		D		
Emulators (3270/5250/ASCII)	D						
Display presentation	D						
Selectable fonts	D						
Code page/national language	D						
Translate PM to X (OS/2)	D						
Data stream interceptor		D					
User-to-program router		D					
Large object				D			
Folders				D			
Electronic document interchange				D			
Access to special-purpose hardware				D			
Event notification (status/triggers)				D			
File system					D		
Database					D		
Workflow manager					D		
Data feed broadcast							D
Download snapshots							D
Regional consolidation							D
Read/write through							D

D = Required distributed function

can support the enterprise distributed solutions desired by customers.

4. The completeness of this common architecture in a given environment such as MVS, OS/2, or AIX is measured by its coverage of the reference design requirements (in Table 1).
5. Since customers' environments are increasingly multivendor, the technology selected for each distributed service in Table 1 must be widely available across vendors' platforms. The common architecture should be open and guided by industry and *de facto* standards.

Acknowledgments

The authors wish to acknowledge the support of Ed Altman and Mike Saranga, who commissioned this effort, and express appreciation for the contributions of the IBM professionals who were part of the work group, including Brian Buckingham, Mike Campbell, Tom Caracio, Mike Cocklin, Don Daria, Cort DeVoe, Roger Harvey, Rob High, John Hildreth, Don Holtz (co-chairman), George Hutfilz, Gene Jurrens, Dave Larkin, Ray Larner, Jay Leiserson, R. Martin, Dave Pullin, Matt Schein, David Turek, Jim Walsh, Shaula Yemini, and Bob Zeliff.

Cited reference

1. A. L. Scherr, "SAA Distributed Processing," *IBM Systems Journal* **27**, No. 3, 370–383 (1988).

Accepted for publication May 18, 1993.

John J. Shedletsky *IBM Client/Server Computing, Route 100, Somers, New York 10589.* Dr. Shedletsky is the Director of Open Client/Server Technology and is responsible for IBM's technical strategy for client/server computing. His previous assignment was Director of Technical Strategy Development, with responsibility for technical strategies in application and system software. Prior to this corporate assignment, he was the program manager for Distributed Systems Architecture, reporting to the Assistant General Manager of Systems Structure and Management in Programming Systems. His responsibility was to define the open IBM product strategy, and to coordinate its implementation across IBM's lines of business. Dr. Shedletsky received his Ph.D. from Stanford University in 1976 and joined IBM at the T. J. Watson Research Center that same year. While in Research he co-invented the EVE logic simulation machine and developed a programmable TIMER in an effort to design 1-cycle System/370* instruction processors. He joined the corporate Engineering, Programming, and Technology staff in 1982. Since then, he has held various design and development management positions at IBM facilities in Austin, Texas, and Kingston, New York. He was the design manager for the first release of AIX and, most recently, the third-line development manager responsible for several IBM 3270 emulation products.

John J. Rofrano *IBM Personal Systems, East Fishkill, Route 52, Hopewell Junction, New York 12533 (electronic mail: IBMMAIL(USIBMUA4) or rofrano@vnet.ibm.com).* Mr. Rofrano is currently a senior programmer and technical lead on the Personal Systems Information Warehouse* development team. He is responsible for the overall architecture, design, and development of the PS Information Warehouse, which is distributed across OS/2 and AIX workstations and servers in a variety of LAN and WAN configurations. He joined IBM in 1984 in the System Products Division in White Plains, New York, as an information center analyst. There he specialized in the support of personal computer products. In 1986 he became manager of the Information Center for the Information Systems and Products Group and a year later became manager of Customer Services I/S at IBM Corporate Headquarters in Purchase, New York. In 1989 he joined IBM's Application Solutions organization, working in the Cooperative Processing Cluster in the Application Systems Division as a technical planner. He was responsible for strategy and plan evaluation of future products that would enable cooperative processing. During this time, Mr. Rofrano served as IBM's representative to the GUIDE Cooperative Processing Project and worked with members of GUIDE, SHARE, and numerous other customers on their distributed application designs. In 1991, he joined the Architecture and Development Technical Staff in Application Solutions where he was one of the chief architects of the Application Services Architecture. He received a B.S. in computer science in 1984 from Mercy College in New York.

Reprint Order No. G321-5527.