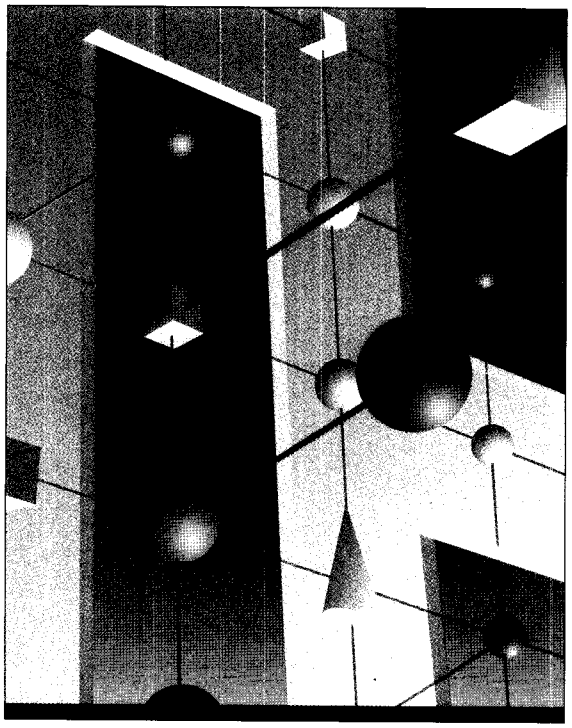
T A N D E M

SYSTEMS REVIEW

VOLUME 10 NUMBER 1

JANUARY 1994



Himalaya Hardware Architecture

Object-Oriented Technology

GDSX

Technical Information and Education

Product Update

T A N D E M

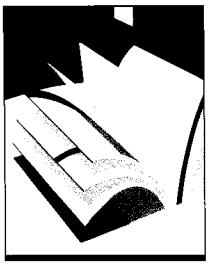
SYSTEMS REVIEW

VOLUME 10, NUMBER 1

JANUARY 1994

The Tandem Systems Review *publishes technical information about Tandem software releases and products. Its purpose is to help programmers, analysts, and other IS professionals to plan for, install, use, and tune Tandem systems.*

TANDEM



Editor's Note

Tandem recently announced the new NonStop Himalaya range of servers, designed to provide massively parallel processing for business applications. The first article in this issue of *Tandem Systems Review*, "A Hardware Overview of the NonStop Himalaya K10000 Server," describes architecture and design features of the K10000, the most powerful and expandable Himalaya server. The article also describes the TorusNet interconnection network, which allows upward scalability from 2 to 4,080 processors.

The second article, entitled "Extending the Client/Server Model With Object-Oriented Technology," examines issues faced by application programmers as the technology supporting client/server computing continues to change rapidly. The article discusses basic concepts of object-oriented technology; describes how it contributes to a more flexible computing environment; and examines how object technology can benefit each stage of application development.

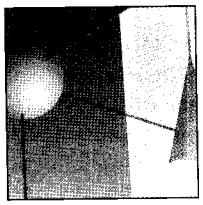
The third article in this issue, "Basic Uses and New Features of Extended GDS," examines the Extended General Device Support (GDSX) product. GDSX simplifies the development of front-end and back-end processes that communicate with I/O devices. This article begins by defining GDSX front- and back-end processes. It then describes GDSX uses in a Tandem Pathway environment and discusses enhancements made available with the D-series releases of GDSX.

This issue continues to provide the quarterly "Product Update" and "Technical Information and Education" departments. The TIE department highlights a new training method, called Audiodigital Technology, being implemented by Tandem Education.

–AL

Tandem Systems Review is published quarterly by Tandem Computers Incorporated. All correspondence and subscriptions should be addressed to *Tandem Systems Review*, 10400 Ridgeview Court, Loc 208-65, Cupertino, CA 95014.

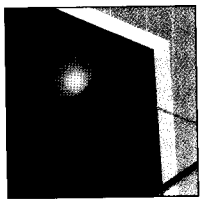
Subscriptions: $75.00 per year; single copies are $20.00. Detailed subscription information is provided on the subscription order form at the end of this book.



HIMALAYA SERVER

FEATURES

DEPARTMENTS



ProductUpdate

Integrity Systems

New Integrity FT Systems

September 1993

Tandem's Integrity FT systems family comprises a range of fault-tolerant, high-performance computing platforms designed for UNIX SVR4 applications that require the highest levels of system availability and data integrity. Tandem now offers three new Integrity FT system models: the CM-1450, the CO-1450, and the S300E.

The CM-1450 and CO-1450 models, based on the MIPS® R4000® RISC microprocessor, can provide more than 100 percent better performance than current Integrity FT systems that are based on the R3000® RISC microprocessors. The number of applications areas that can be addressed by the Integrity FT product family is now expanded.

The base configuration of the CM-1450 includes 64 megabytes of local memory and 16 megabytes of global memory, four 1-gigabyte disk drives, two SCSI controllers, and one service processor. Maximum local memory on the CM-1450 has been increased to 128 megabytes; maximum total memory is still limited to 192 megabytes.

The model CO-1450 incorporates the same specialized features for installation in telecommunications central office environments as the model CO-1300. These features include compliance with stringent safety, fire resistance, earthquake resistance, temperature, power, and grounding standards.

The CM-1450 and CO-1450 models feature the same system cabinets and mass storage cabinets as the CM-1300 and CO-1300 models respectively. Up to 9 disk or tape devices can be stored in the CM-1450 or CO-1450 system cabinet, while up to 21 disk or tape devices (per cabinet) can be stored in up to two optional mass storage cabinets.

The model S300E is a new entry-level Integrity FT system that replaces the model S100E. The S300E is a higher performance entry-level system that provides all the benefits of the Integrity FT architecture at a price under $100K.

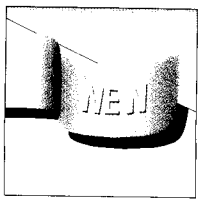
Integrity NR/4001 Server

August 1993

The Integrity NR/4001 is a compact, high-performance, single-processor server designed for small or medium-sized workgroups. Although the NR/4001 is as small as a personal computer, it provides the power, memory, and storage capacity of a much larger system.

The base model of the Integrity NR/4001 includes a 100-MHz R4000SC processor (upgradable to the 150-MHz R4400SC processor), 32 megabytes of memory (expandable to 192 megabytes), and a 1-gigabyte internal system disk that contains the IRIX™ operating system and the NFS software. The disk capacity can be expanded to 15 gigabytes by adding user disks.

Other features of the NR/4001 include 4 industry-standard 32-bit EISA slots or 2 EISA and 2 64-bit GIO64 slots, 2 SCSI-2 channels, internal and external tape and CD-ROM options, an integrated Ethernet subsystem with both 10BaseT and AUI ports, and support for 1 or 2 additional Ethernet or FDDI communications adapters. An NR/4401 disk expansion cabinet holds up to 6 3.5-inch peripherals.



The Product Update department provides brief descriptions of new products announced by Tandem. For more information on any of these products, please consult your local Tandem representative.

Integrity NR/4412 Server

August 1993

The Integrity NR/4412 is a midrange, cost-effective SMP deskside server that offers scalable performance and has a highly expandable storage capacity. The NR/4412 has two base models including either two or four 150-MHz R4400MC processors; 64 megabytes of ECC, coherent shared memory (expandable to 2 gigabytes); a QIC tape drive; a CD-ROM drive; and a 2-gigabyte internal system disk that contains the IRIX operating system and the NSF software.

Other features of the NR/4412 include: from 2 to 12 SMP R4400MC processors (150 MHz); parity-protected POWERpath-2 system bus (1.2 gigabytes/second); from 1 to 3 POWERchannel-2 bus slots (320 megabytes/second); 5 VME64 industry-standard bus slots; as many as 24 SCSI-2 channels; and up to 720 gigabytes of internal disk storage and 2.5 terabytes of external RAID disk storage. An NR/4412 disk expansion cabinet can house up to 6 3.5-inch peripherals.

Integrity NR/4436 Server

August 1993

The Integrity NR/4436 is a high-end, symmetrical multiprocessing server designed to support enterprise-wide distributed computing environments. Along with its exceptional scalability and very large storage capacity, the NR/4436 is the most powerful server in the Integrity NR series.

The Integrity NR/4436 has two base models with either two or four 150-MHz R4400MC processors; 64 megabytes of ECC, coherent shared memory (expandable to 2 gigabytes); a QIC tape drive; a CD-ROM drive; and a 2-gigabyte internal system disk that contains the IRIX operating system and the NFS software. The NR/4436 can have up to 960 gigabytes of internal disk storage and 2 gigabytes to 3.5 terabytes of external RAID disk storage.

Other features of the NR/4436 include: from 2 to 36 SMP R4400MC processors; parity-protected POWERpath-2 system bus (1.2 gigabytes/second); from 1 to 4 POWERchannel-2 bus slots (320 megabytes/second); from 5 to 25 VME64 industry-standard bus slots (and support for additional Ethernet and FDDI); as many as 32 SCSI-2 channels; as many as 4 HIO buses (and support for FDDI and HiPPI); and a built-in Ethernet controller. The NR/4436 disk expansion rack supports 7 SCSI bays, each holding up to 8 3.5-inch or 5.25-inch SCSI devices.

Workstation and Terminal Products

Indigo R3000 and R4000 Workstations

August 1993

The Indigo® R3000 and R4000 workstations are high-performance client platforms for demanding networked applications. Each model features a compact mini-tower enclosure; 3 SCSI-2 connectors for internal devices and one connector for external devices; 2 RS-422 serial ports and one parallel port; an integrated Ethernet subsystem; and a 16-inch or 19-inch noninterlaced color monitor with 1024 x 768 or 1280 x 1024 resolution.

The R3000 workstation uses a 33-MHz MIPS RISC R3000A processor (upgradable to the 100-MHz R4000SC or the 150-MHz R4400SC) and supports up to 96 megabytes of memory. The R4000 uses a 100-MHz MIPS RISC R4000SC processor (upgradable to the 150-MHz R4400SC) and supports up to 384 megabytes of memory. Both models have an internal disk capacity of up to 2 gigabytes.

Indigo workstations comply with the MIPS System V Application Binary Interface (MIPS ABI). Applications that comply with the MIPS ABI can run without recompilation on Indy™ and Indigo workstations, Tandem Integrity NR and Integrity FT servers, and other UNIX systems that conform to the MIPS ABI. Indigo workstations and Integrity NR servers also share a single version of the IRIX 5.x operating system, the most advanced, user-friendly implementation of the UNIX operating system available today.

Indy Workstation

August 1993

The Indy workstation is a low-cost, high-performance desktop computer featuring large memory and storage capacity, integrated networking, graphics, and digital media. It performs at much higher levels than Pentium-powered PCs, with better system throughput and more advanced integrated digital media capabilities. In addition, the Indy offers the industry's first media user interface, Indigo Magic, as a standard package.

The Indy workstation uses a 100-MHz MIPS R4000PC or R4000SC processor, supports up to 256 megabytes of memory, and has an internal disk capacity of 2 gigabytes. Other features include: a 400-megabyte/second (burst rate) memory bus; 2 internal SCSI-2 connectors for internal devices and 1 connector for external devices; 2 RS-422 serial ports and 1 parallel port; an integrated 15DN, AUI, or 10BaseT Ethernet subsystem; color monitors in 15-inch, 16-inch, and 19-inch sizes, with 1024 x 768 or 1280 x 1024 resolution; line-level analog stereo and serial digital stereo; ISDN integrated into the system; a floptical drive; support for PC and Macintosh network software and third-party PC and Macintosh emulation packages; and integrated video ports.

A Hardware Overview of the NonStop Himalaya K10000 Server

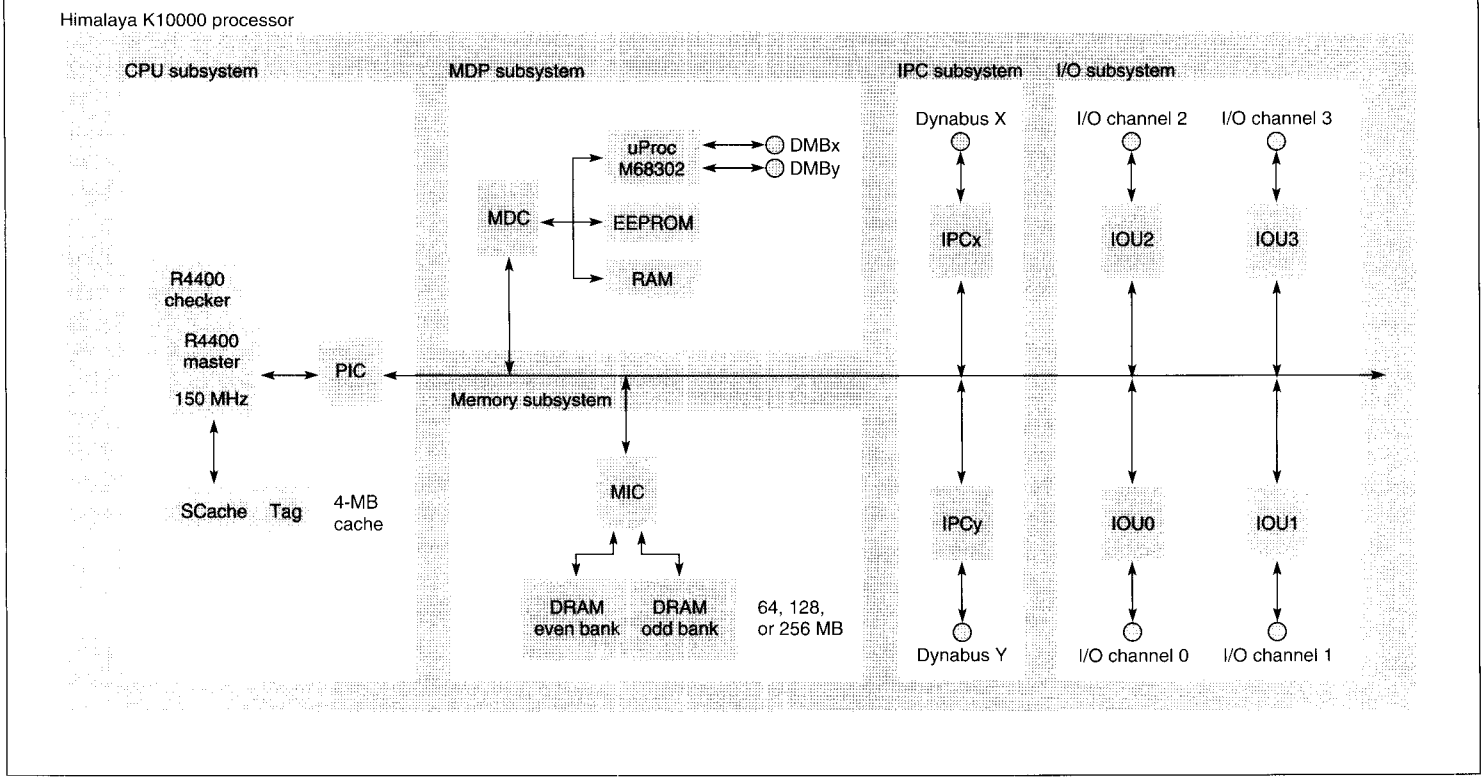
With recent advances in technology and with falling hardware costs, commercial data processing applications are moving increasingly toward online processing. These applications, which perform tasks such as online transaction processing (OLTP), database manipulation, messaging, and decision support, require continuous system availability, data integrity, secure operations, and a distributed processing environment.

The Tandem™ NonStop™ Himalaya™ servers, based on the loosely coupled, multiprocessor architecture of previous Tandem systems, provide massively parallel processing that can meet the technical and economic requirements of these applications. The K10000 server is the most powerful and expandable member of the Himalaya product family. It can scale upwards from 2 to 4,080 processors to handle large business applications. This scalability is achieved through the TorusNet™ interconnection hierarchy.

The K10000 server constitutes a new generation of the Tandem NonStop Series/RISC (TNS/R) systems. (The articles by Faby and Mateosian, Blanchet, and Cressler in the Spring 1992 issue of the *Tandem System Review* give detailed descriptions of TNS/R systems.) By combining RISC technology with the Tandem NonStop Kernel operating system (Bartlett, 1981), the K10000 server provides the foundation for a smooth migration to future advances in technology. The server also maintains complete compatibility with all applications running on existing Tandem systems.

This article gives an overview of the Himalaya K10000 server's hardware architecture. It describes design features and practices that significantly improve the server's data integrity, performance, and reliability. In addition, it discusses the TorusNet connectivity architecture, which further improves the server's ability to provide efficient linear growth.

Figure 1



Overview of the K10000 Hardware Architecture

The K10000 processor contains five subsystems: the central processing unit (CPU) subsystem, memory subsystem, interprocessor communication (IPC) subsystem, I/O subsystem (IOS), and maintenance and diagnostic processor (MDP) subsystem. Figure 1 is a diagram of the K10000 processor.

CPU Subsystem

The CPU subsystem is the main processing unit of the K10000 processor. It contains a pair of industry-standard MIPS® R4400™ RISC processors, four megabytes of cache, and a high-density, 1-micron CMOS application-specific integrated circuits (ASIC) processor interface chip (PIC).

The R4400 is a 64-bit RISC processor employing superpipeline architecture. It has on-chip, first-level 16-kilobyte caches for both instructions and data. A 4-megabyte off-chip, second-level cache is implemented to take full performance advantage of the RISC processor. In addition, Tandem guarantees data integrity for the R4400 RISC CPU's operations by using a pair of R4400s to implement lock-stepped, master-checker checking. On every R4400 CPU clock, which runs at a speed of 150 MHz, the master and checker R4400 RISC processors perform exactly the same operation (that is, they are lock-stepped together). Their outputs are compared with each other to ensure correct operation. Any inconsistent comparison due to CPU hardware failures will lead to a processor halt. If a processor halt occurs, the MDP subsystem can collect the processor status and send it to a remote processor to determine the cause of the error condition.

Figure 1.
Logical block diagram of a Himalaya K10000 processor.

Memory Subsystem

The memory subsystem provides the K10000 processor with main memory configurations of 64, 128, or 256 megabytes. It consists of a memory interface chip (MIC) ASIC and two banks of industry-standard dynamic random-access memory (DRAM) arrays. The MIC provides all the necessary capability to manage the DRAM arrays. The memory subsystem is protected through single-bit error correction and multiple-bit error detection code.

IPC Subsystem

The IPC subsystem links multiple processors together to form different system configurations. It provides two high-speed buses, the Dynabus™ X and Dynabus Y, to form dual pathways among processors. Each IPC ASIC is responsible for one Dynabus and implements an independent direct memory access (DMA) engine to handle data transfer between the processor itself and other processors. The DMA engine moves data directly in and out of the memory subsystem for interprocessor communication, without the assistance of the CPU. This relieves the CPU for other processing tasks, so more useful work can be achieved on the CPU during Dynabus data transfers between processors.

I/O Subsystem

The I/O subsystem (IOS) is the conduit for the K10000 processor to communicate with I/O devices such as disks and local area networks (LANs). The K10000 processor can have either two or four Tandem I/O channels, one channel per IOS ASIC, each with its own microcoded DMA engine connecting it to the memory subsystem.

The NonStop Cyclone™ processor also provides up to four I/O channels, but each pair of I/O channels shares a DMA engine to the processor. The new I/O structure on the K10000 results in a 100-percent improvement over the NonStop Cyclone in aggregate I/O bandwidth. With this new K10000 I/O capability, users can configure their I/O structures to maximize performance or minimize cost, giving them flexibility in meeting their needs.

MDP Subsystem

The MDP subsystem provides maintenance and diagnostic functions for the K10000 processor. It is based on the maintenance and diagnostic subsystem architecture developed for the NonStop VLX™ and Cyclone systems (Allen and Boyle, 1987). It provides a pair of maintenance buses, DMBx and DMBy, to link to a system-level, fault-tolerant maintenance and diagnostic subsystem. The MDP subsystem is also responsible for initializing the processor and performs failure-data collection when an error is detected on the main processor.

Performance

The K10000 processor achieves more than twice the performance of the NonStop Cyclone processor (Horst, Jardine, and Harris, 1990) in about one-fourth to one-sixth the printed wiring board (PWB) space (when memory expansion is accounted for). That is, the K10000 processor, which resides on one PWB, provides twice the performance of the Cyclone processor, which resides on four to six PWBs. This improvement is achieved by using RISC technology coupled with high-density CMOS ASICs, as described previously. By improving performance and lowering processor costs, the K10000 forms the foundation for providing a wide range of cost-effective product solutions for Tandem users.

Reliability and Maintainability

The proven architecture of Tandem systems has provided reliable computing for a wide range of commercial applications. The K10000 is built on this foundation and further advances it. To improve the quality of the end product, the K10000 product development team deployed many design practices such as highly accelerated life testing (HALT), qualifying suppliers and new components, and root cause analysis.

HALT uses an environmental test chamber that can apply extreme environmental conditions such as high temperature, high humidity, and high electrical power supply voltage to simulate the life cycle of a product in a relatively short time period. It provides an accelerated, continuous-improvement environment to keep users from experiencing quality and reliability failures.

The development team adopted standard methods (such as vendor visits, statistical process monitoring, and component construction analysis) for qualifying suppliers and new components for the K10000. This vigorous qualification and monitoring process improves the quality of Tandem hardware by continuously monitoring vendor capability and eliminating substandard suppliers and components.

The K10000 processor achieves more than twice the performance of the NonStop Cyclone processor.

When design defects were uncovered during the development of the K10000 (from the design phase through the manufacturing phase), the development team performed root cause analysis. Using the results of their analysis, the team took corrective actions to ensure that defects were eliminated as well as to prevent the same or like defects from recurring.

These efforts resulted in a significant reduction in the predicted corrective maintenance rate for the K10000 (compared with previous Tandem systems).

Figure 2

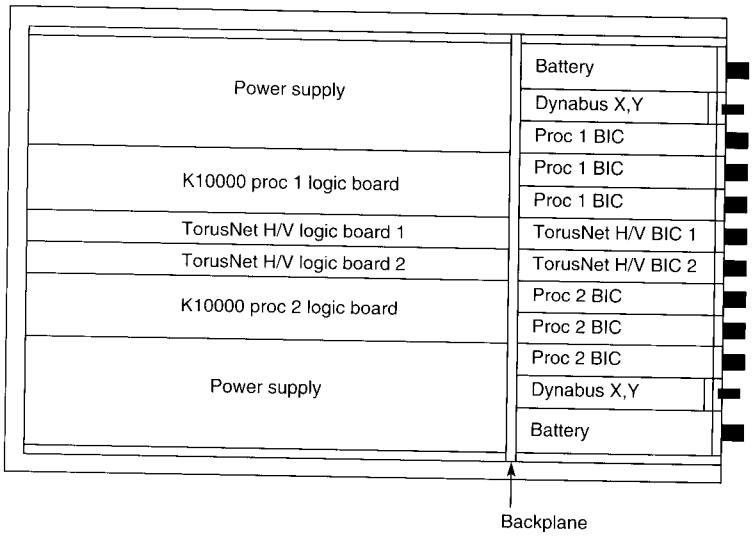


Figure 2.
Top view of the Himalaya processor cabinet. (BIC = backplane interconnect card.)

Figure 3

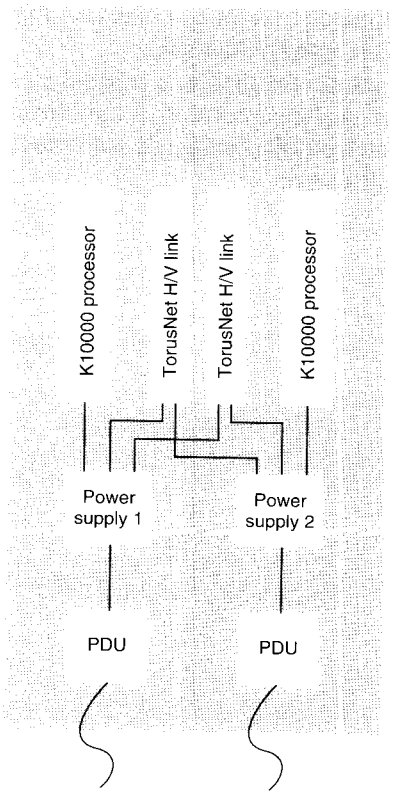


Figure 3.
Dual power distribution units (PDUs).

K10000 Power Architecture

The approach to power distribution on the K10000 differs from that of previous Tandem systems. Figure 2 gives a top view of the K10000 processor cabinet. As Figure 2 shows, the power supplies are plugged into the backplane directly from the two outside slots. This structure not only makes the power distribution to the processor logic simpler, it also simplifies construction and hence lowers the cost of the system.

As shown in Figure 3, the dual AC power distribution units inside the processor cabinet are powered by two separate power cords. With separate external power wiring, loss of a single power source or power cord will not bring down the whole processor cabinet. Thus, separate power cords provide better system availability than is possible with a single power-cord system.

Service Strategy

In keeping with traditional Tandem system architecture, almost all units inside the K10000 processor cabinet are field-replaceable units (FRUs). In a K10000 processor cabinet, all units are FRUs except the backplane and the card cage. The granularity of the serviceable units on the Dynabus is further divided to allow each X or Y bus to be serviced independently, without affecting the other bus. Thus, one can service the Dynabuses without affecting the system operation.

System Expandability

The K10000 server can scale upwards from 2 to 4,080 processors to meet the needs of any large commercial application. This massively parallel processing (MPP) architecture is achieved by using the hierarchical, scalable TorusNet interconnection network. With TorusNet, users can select a cost-effective, optimal configuration of K10000 processors to suit their application requirements.

Figure 4

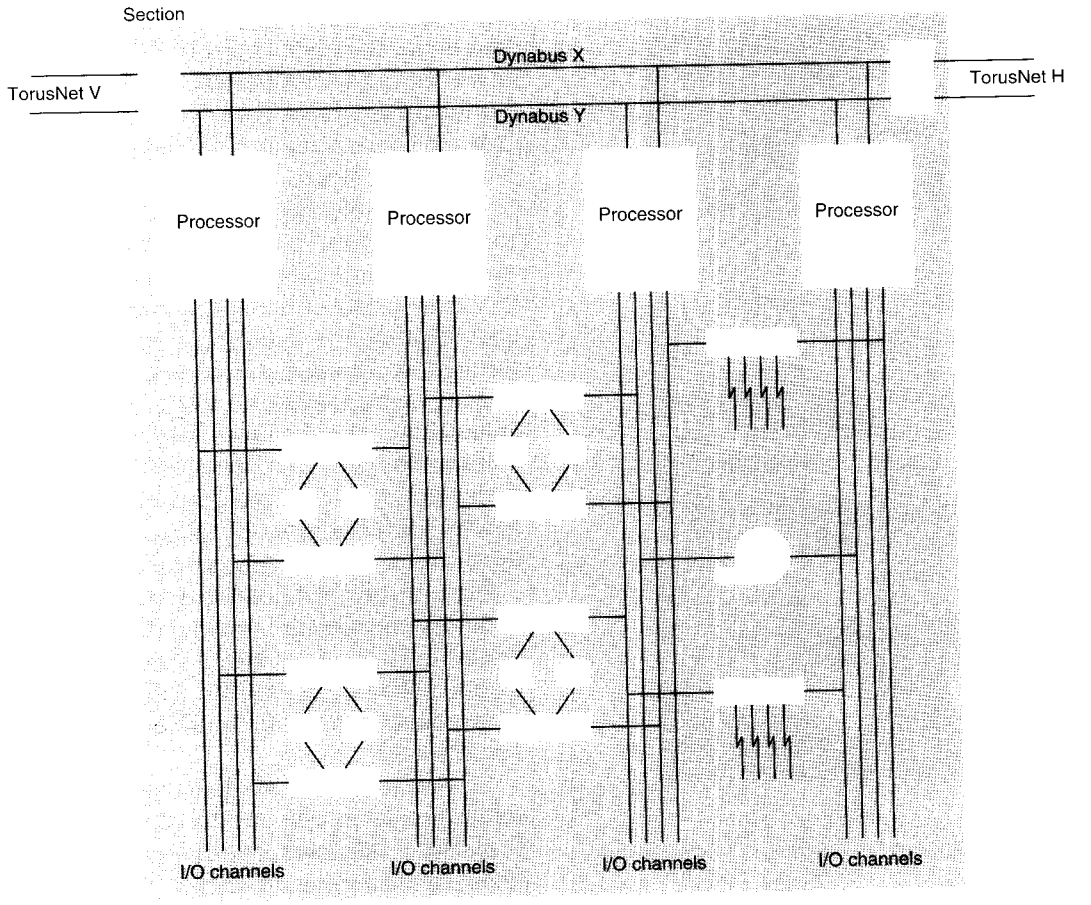


Figure 4.
A logical view of the TorusNet architecture: a section containing four cells. TorusNet extends the interprocessor bus beyond one section.

The TorusNet network for the K10000 server consists of *cells*, *sections*, *nodes*, and *domains*. A K10000 cell, the basic element in the network, is a K10000 processor with its associated memory, I/O channels, storage elements, and communication devices, as shown in Figure 4.

A section contains from two to four K10000 cells. The K10000 cells within a section are interconnected through the Dynabus X and Dynabus Y (shown in Figure 4). Multiple sections can be interconnected through two-dimensional TorusNet H and TorusNet V fiber-optic connections to form a TorusNet node or domain, as shown in Figure 5.

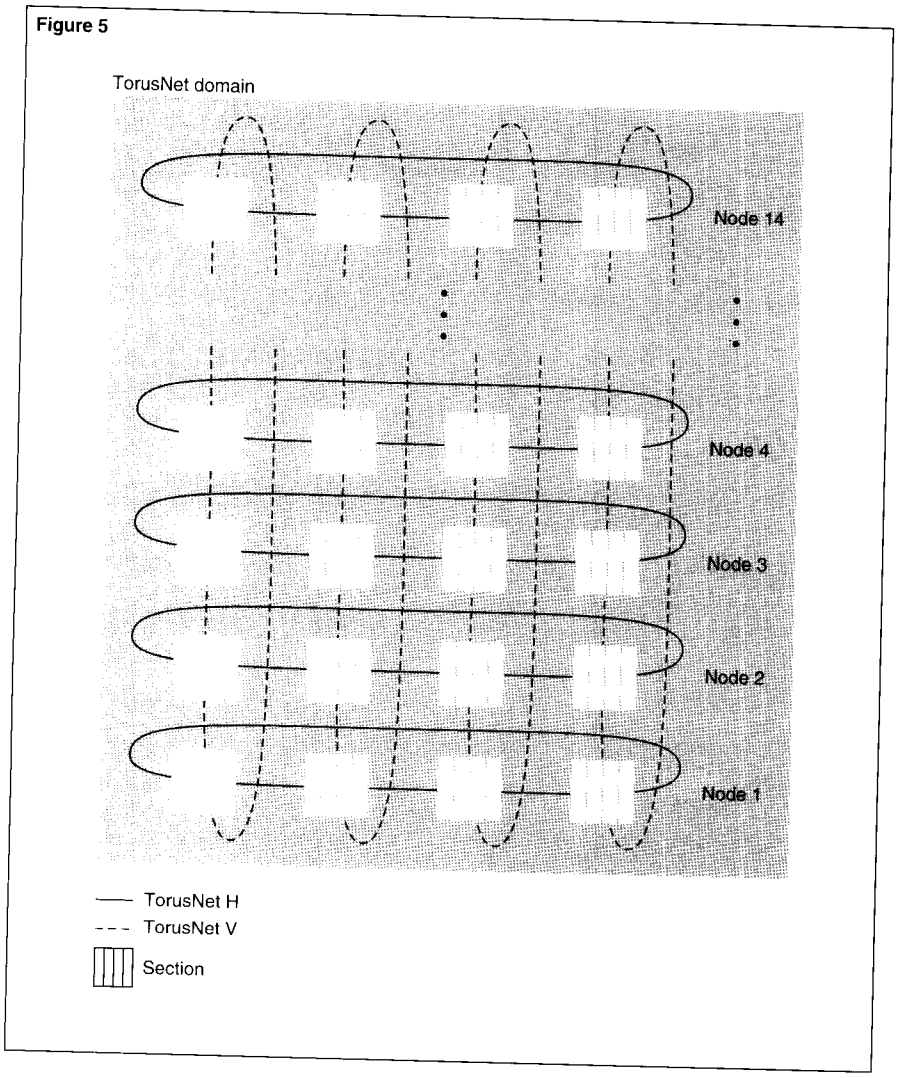


Figure 5.
A logical view of the TorusNet architecture. A node can contain up to four sections. A domain can contain up to 14 nodes (56 sections).

The TorusNet H links and TorusNet V links together form a two-dimensional interconnection network. In one dimension, the TorusNet H links form a circular ring connection, which can connect up to four sections together to form a K10000 server node. The TorusNet V links form the other interconnection dimension and can connect up to 14 nodes in a circular ring structure to form a domain. Thus, each section within a domain has four links for data communication, two in the TorusNet H link dimension and two in the TorusNet V link dimension. Using both the TorusNet H and TorusNet V connections, one can connect up to 224 K10000 cells within a TorusNet domain.

The TorusNet H provides functions similar to the Dynabus+™ connection available on previous Tandem systems such as the NonStop Cyclone system. A K10000 server node is equivalent to a 16-processor system connected through the Dynabus+. However, the FOX™ fiber-optic ring mimics a single TorusNet V connection, whereas up to four TorusNet V connections exist in a TorusNet domain. The scalable TorusNet H and TorusNet V interconnections offer several advantages over traditional Tandem interconnection networks.

First, a TorusNet domain allows for growth in interconnection data bandwidth on both the TorusNet H and TorusNet V links. Each new section added to the domain will add up to four more connections, two TorusNet V and two TorusNet H.

Second, with the two-dimensional torus connection formed by the TorusNet H and TorusNet V interconnections, processors can exchange data more efficiently. If the data transfer is outside a node, the data is first routed through the appropriate TorusNet V interconnection. (If the data transfer is within a node, the data is routed through the TorusNet H links associated with the node.) When the data reaches the destination node, it travels through the TorusNet H interconnection to reach its intended destination.

The third dimension of the TorusNet, called the TorusNet D interconnection, forms the communication link among domains. It can link TorusNet domains to form systems containing up to 4,080 K10000 processors. This interconnection uses the Expand™ data communications network available on previous Tandem products.

Conclusion

The K10000 server enhances the traditional Tandem NonStop system architecture to accommodate the emerging massively parallel processing paradigm for commercial applications. The K10000 is completely compatible with most existing I/O and communications peripheral devices and with all Tandem application software. With this compatibility, users can easily migrate their existing Tandem hardware and software to the K10000 server. In addition, the TorusNet interconnection network allows users to expand the system cost-effectively as their applications grow.

References

Allen, J. and Boyle, R. 1987. The VLX: A Design for Serviceability. *Tandem Systems Review*. Vol. 3, No. 1. Tandem Computers Incorporated. Part no. 83939.

Bartlett, J. 1981. A NonStop Kernel. *Proceedings of the Eighth Symposium on Operating System Principles.*

Blanchet, M. 1992. Improving Performance on TNS/R Systems with the Accelerator. *Tandem Systems Review*. Vol. 8, No. 1. Tandem Computers Incorporated. Part no. 65250.

Cressler, D. 1992. Debugging Accelerated Programs on TNS/R Systems. *Tandem Systems Review*. Vol. 8, No. 1. Tandem Computers Incorporated. Part no. 65250.

Faby, L. and Mateosian, R. 1992. Overview of Tandem NonStop Series/RISC Systems. *Tandem Systems Review*. Vol. 8, No. 1. Tandem Computers Incorporated. Part no. 65250.

Horst, R., Jardine, R., and Harris, R. 1990. Multiple Instruction Issue in the NonStop Cyclone Processor. *Seventeenth International Symposium on Computer Architecture*. Also *Tandem Technical Report* 90.6. 1990. Tandem Computers Incorporated. Part no. 48007.

Cheng Kong is the hardware project leader for the NonStop Himalaya K10000 processor development. Before joining Tandem in 1987, he worked for six years designing RISC processors in the computer industry. Cheng holds a Ph.D. in computer engineering from SUNY Buffalo.

Extending the Client/Server Model With Object-Oriented Technology

The demand for client/server computing continues to grow as its benefits become known: reduced costs, increased performance, enhanced user productivity, and, most important, widespread access to corporate information formerly contained within the domain of proprietary mainframe systems. With client/server applications, users can manipulate corporate information with easy-to-use software running on their PCs or workstations.

Quickly changing technology, which has made client/server computing possible, also presents the greatest challenge to information systems (IS) managers and developers. Current client/server solutions may be superseded by new technology and standards. Moreover, users will continue to think of new uses and services to be built on enterprise data. In response, IS professionals have to create a computing environment flexible enough to satisfy current and future market demands and adaptable enough to endure the next paradigm shift in technology.

To avoid having to rewrite or rework applications each time a new technology is introduced, IS professionals must create a well-defined, modular client/server environment. In particular, they must:

- Build a framework that hides technology and encapsulates business logic so that the application is not locked into using a unique or proprietary product or tool.
- Create common application services and functions to be treated as reusable software components.
- Maintain a consistent client/server development methodology.

Object-oriented technology (henceforth referred to as object technology) can help IS professionals achieve these goals. Object technology is an integrated approach to application analysis, design, and programming that simulates real business processes rather than dividing the business problem between data and function. With an object-oriented programming language such as the Tandem™ C++ language, programmers can create reusable and extensible building blocks that can be used for application development by other programmers who use C++ or other languages.

This article begins by defining two client/server models. It then discusses the issues: achieving technology independence, creating reusable services, and promoting a consistent development methodology. Next, it introduces some basic concepts of object technology. (To explore these concepts further, readers should refer to the bibliography at the end of the article.)

The article then examines how object technology can benefit each stage of application development, from analysis to implementation. It addresses the issues listed above as they appear in the application life cycle. It shows how object technology extends the client/server model, providing a clear role for a powerful client and server.

The article assumes readers are familiar with client/server architecture. It is intended for all IS professionals, including managers who choose technology and developers who code and implement it.

The Client/Database Server and Client/Transaction Server Models

In most current client/server solutions, a client application running on a workstation retrieves data from a host where the database management system (DBMS) resides. Typically, a gateway translates the data so that host and client can communicate. With this model, called the client/database server model, applications are easy to develop. The model also offers perceived ease of enhancement. Users achieve these benefits by acquiring tools that create client applications and require little or no coding of servers.

This computing model is used predominantly by a relatively small set of users for decision support or departmental applications. This model only partly satisfies users' objectives. Users also want to:

- Expand the use of the tools they already have on their PCs.
- Use their PCs to gain better access to global enterprise data (usually found on mainframes or superservers).
- Have IS professionals act as consultants to help them achieve their goals, but be able to design and even build application functions themselves.

To meet the increasing demand for high-performance, robust client/server applications, IS professionals are turning to a different model, one that comes closer to supporting genuine distributed computing. The model has several names: client/transaction server, client/server OLTP, and distributed function.

The client/transaction server model divides business-application functions between two or more computer systems. It places the graphical user interface (GUI) on the workstation and the DBMS on one or more systems. By splitting the business logic between client and server, it distributes functions to the best resource, thus providing high performance.

As users demand increasingly complex functions, these applications must also offer high availability, fault-tolerance, scalability, and integrity. Client/transaction server applications can provide these benefits.

Definitions

Abstract data type (ADT): a combination of a data structure and operations on that data. ADTs are similar to the data types built into a programming language, but they are not part of the language definition. Programmers create ADTs as object classes.

Application middleware: a host-language interface or application programming interface (API) that transmits requests between local and remote applications. In particular, a consistent set of verbs used by developers to initiate requests to remote applications.

Class: a description of a set of similar objects. A class provides the template for creating individual objects, each of which is an instance of the class.

Encapsulation: the technique of combining data and code into an object class. Encapsulation hides information and shields the complexities of the object's behavior. It allows other system components to be insulated from the implementation details of the object's operations.

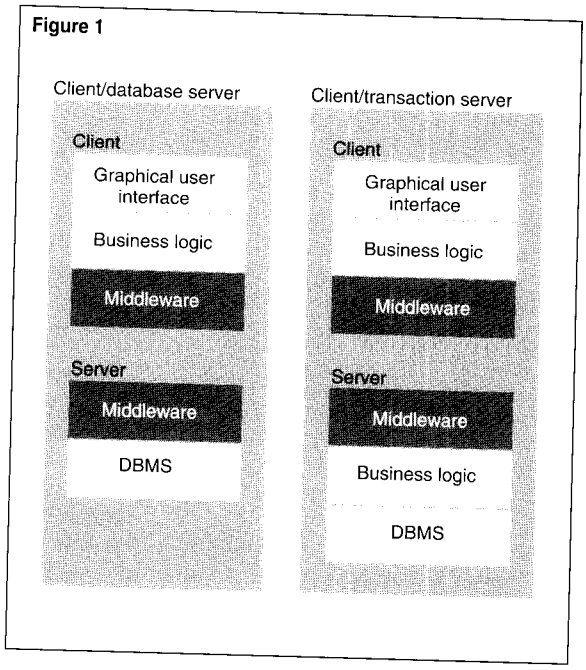
Framework: a specification or implementation (that is, a set of classes) that solves some application problem or requirement. In a client/server environment, a framework provides a set of common technical functions that can be combined with business logic to create a complete solution.

Object: a concept, method, or software module that encapsulates related state and behavior. The state is represented by data and behavior is represented by procedures that modify the data (change the state).

Object-based: pertaining to any method, language, or system that supports object identity, classification, and encapsulation.

Object-oriented: pertaining to a method or language that is object-based and supports specialization (inheritance).

Figure 1.
Layers of functional responsibility in client/server applications.



However, building complex client/server applications is challenging. IS professionals must be able to change and enhance the applications quickly. They must also choose which *application middleware* technology to use. Application middleware (henceforth referred to as middleware) is the software used by clients to make requests of servers. Examples are Tandem's Remote Server Call (RSC) software, the Remote Procedure Call (RPC) component of the Distributed Computing Environment (DCE) standard, IBM's Common Programmer Interface-Communication (CPI-C) software, the Common Object Request Broker standard, and the Application-Transaction Manager Interface (ATMI) component of the TUXEDO transaction processing (TP) monitor.

These issues can create uncertainty among IS professionals. Each investment in a particular technology involves a risk. The remainder of this article discusses these risks in more detail and describes how object technology can help to reduce them.

Technology Lock-in

An application is locked into using unique or proprietary products (software or tools) when those products become an intrinsic part of the solution. In particular, lock-in occurs when one allows the application middleware's API to become tightly intertwined with the code that performs business logic or manages the user interface. An application designed in this way contains code that is difficult to migrate or redesign when solution providers want to integrate new or different technologies.

The term *proprietary* refers to products controlled by a single vendor or limited group of vendors; users can acquire them only from those vendors. For example, most application middleware APIs, even those implemented by more than one vendor, can be proprietary.

Monolithic Architecture

Technology lock-in can occur when users build applications in a monolithic fashion. Monolithic refers to applications in which developers designed and implemented code, either as a single module or as multiple interacting modules, that did not separate functions into distinct layers. In those applications, developers tightly intertwined some combination of the user interfaces, data access, and business functions. Other functions such as data communications and remote data access were mixed into this bowl of spaghetti code.

It took major rewrites to integrate new technology into these applications. Companies expended costly resources to migrate to new environments or take advantage of paradigm shifts such as client/server computing. After evaluating the cost of rewriting applications to adapt to these advances, companies often delayed making any changes.

These constraints force companies to go on supporting and maintaining many legacy applications. They also prevent companies from taking advantage of the client/server model.

Client/Server Architecture

Client/server architecture should differ greatly from the traditional architecture of proprietary mainframe applications. To implement a client/server application, one should separate application functions into four distinct layers:

- User interface (or graphical user interface).
- Business logic.
- Application middleware.
- Resource management (in particular, database access).

Figure 1 shows a physical model of these layers. The client/database server model requires four functional layers. The client/transaction server model requires five layers, because it splits business logic between the client and server(s).

The middleware component logically as well as physically connects the client to the server. In the client/database server model, the middleware may also have a database gateway as one of its components. In the client/transaction server model, the middleware usually contains mechanisms for finding and communicating with individual server processes, which perform the requested services.

As indicated earlier, the client/database server model offers perceived benefits such as integrated tool sets and other technologies that alleviate the need for most programming. These technologies are appealing because they are easy to use, and one can build applications quickly with them.

However, as shown in Figure 2, integrated tools allow and even promote the migration of their APIs across functional boundaries. Usually this is accomplished by using scripts that allow the intermixing of the proprietary API with the user interface provided by the tool. This approach blurs the functional layers, creating a model that resembles a monolithic design.

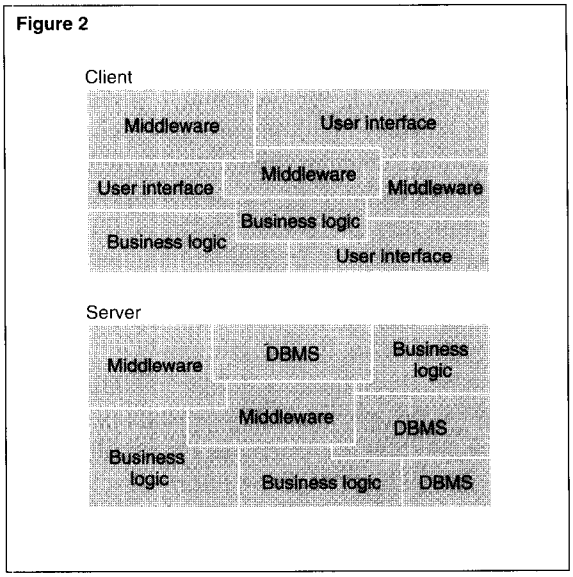


Figure 2.
Integrated tool sets blur the layers of functional responsibility.

Companies migrating their applications to the client/server model should consider the costs as well as the benefits of using integrated technologies. In many cases, companies that allow the boundaries between functions to become fuzzy will face the same problems they faced with their monolithic applications. Some problems associated with integrating new technology may even be compounded by the complexities of distributed computing.

To help solve these problems, users need a methodology that allows the functional layers to be encapsulated, so that each layer is unaffected by a change in the implementation of any other layer. Object technology, discussed later in the article, offers such a methodology.

Reusable Services

Using reusable software components to perform common services increases programmer productivity and improves the reliability of the services. Object technology can help developers achieve this goal in spite of the pressures of constrained budgets and tight development schedules. However, using existing methods (described later), developers often create the same services over and over.

Common services are software modules and subsystems that perform similar functions both within and across applications. Practical constraints force most development groups to prioritize these functions, which should be implemented as common services. These services fall into three categories: critical, required, and desirable.

Critical services, such as security, are created because of their high priority. Development groups plan for them and assign resources to code them. Often, however, developers create these services to be unique to a particular application. Thus, many different implementations of the same service may exist within a company.

Required services perform functions such as error handling, logging, instrumentation, and print handling. Developers usually write these services, but on an ad hoc basis. They implement the services differently and more than once, not only within the same company but often within the same application.

Desirable services include functions such as naming, user event delivery, version control, file transfer, and help. Developers treat these services like documentation. Everyone wants to have them, but usually there isn't time to implement them.

Developers usually use *morphallaxis* methods to implement those common services that do get written. Morphallaxis refers to existing code that is copied to a new file and modified to fit its new purpose. No links are established between the various copies of code. If someone discovers and repairs a bug in the original code, knowledge of this change may or may not reach the person responsible for the new copy of the code. This results in user-found bugs and an increase in maintenance costs.

There are two requirements for creating reusable software. First, developers should package behavior into basic modules that can be tested and documented individually. Second, they must have an acceptable way to extend those modules without modifying or affecting the integrity of the basic modules. Object technology addresses both requirements.

Inconsistent Development Methodology

Traditionally, development groups have used one of two software development methodologies, function-centered or data-centered. In function-centered analysis and design, designers examine the requirements specification for high-level functions. When found, the functions are extracted and defined. Designers then decompose the high-level functions into smaller ones. (This process is sometimes called top-down structured design.) Designers group major functions into applications. As they decompose these functions, they discover required data. In the end, they create data tables.

Function-centered development tends to create processes highly specific to the business problem. These processes perform well but may become difficult to enhance and maintain.

In the data-centered model of analysis and design, developers first find the data attributes that fulfill the requirements of the business problem. They then take this data through some kind of logical and physical design. During this process, they discover the functions that will operate on the data to fulfill the business-problem requirements.

Clearly, both methodologies (data-centered and function-centered) use a process that isolates data from function. A tenet of both methodologies is to achieve data independence by isolating and separating data from procedures. The goal of data independence is to ensure that the structure of information is unaffected by the way it is used.

However, since information access and manipulation is not usually restricted to a set of common operations, the information structure (schema) must be made available to programmers to create operations as needed. Thus, information integrity is dependent on each programmer and has proven to be risky.

The need for referential integrity, either programmed or programmatically supplied in the API, is a consequence of this approach. Without referential integrity, there is no guarantee that table and data associations will be maintained. With this approach, not only does data not become independent, but data integrity becomes difficult to maintain.

The client/server environment can further complicate these development methodologies. First, designers often incorrectly make the physical split between client and server in the analysis or early design phase. Usually this is caused by viewing the business problem exclusively in terms of data or function; designers determine where either the data or the function currently resides and where it should be placed. Among other problems, this split forces the emphasis on technology and away from the business problem.

Second, different groups usually develop the client and server; they use different methodologies and emphasize areas other than the business problem. When the client group does analysis and design, it places primary focus on the user interface. The business problem is treated as a secondary issue. The server group tends to analyze and design using a data-centered view. Its members are concerned with creating a highly optimized database. This group also treats the business problem as a secondary issue.

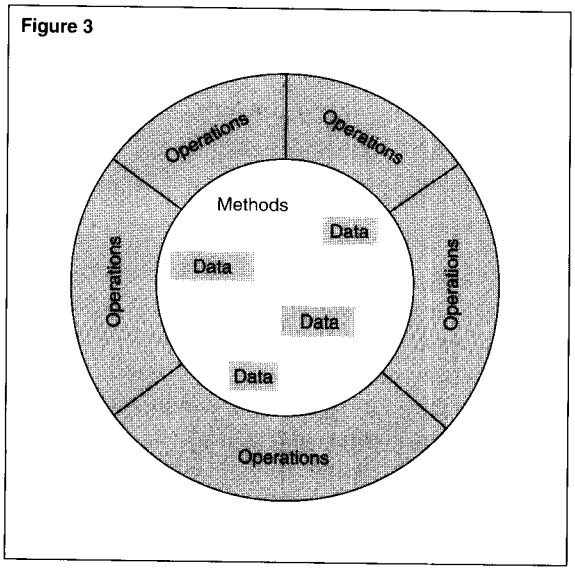
When developers test the client and server, problems are likely to occur. These problems relate not only to maintenance and extendibility of code, but also to an incompatibility of development work, starting with analysis and design and continuing through implementation and the rest of the application life cycle.

The development methodologies described above emphasize either data or function. Neither emphasis greatly assists in creating reusable and extendible software. Nor do these methodologies contribute to the layering and protection of business functions and data. Clearly, a better methodology is needed to address these three related issues.

An Overview of Object-Oriented Technology

Object technology can help extend the client/server model to address the issues of technology independence, reusability, and consistent development methodology. To understand the object-oriented paradigm, one must examine the concepts of object technology. (Though the principles of object technology are consistent, the terminology is not. The terms and definitions in this article are derived from several sources. For more information, refer to the bibliography at the end of the article.)

Figure 3.
Isolating the internal workings (methods and data) of an object. Operations provide external access to the object's behavior.



The key concepts of object technology are *objects*, *classes*, *inheritance*, *polymorphism*, and *dynamic binding*. The first three concepts apply to analysis and high-level design; the last two apply to low-level design and implementation. All of these concepts are discussed in the following sections.

Objects

In object technology, systems and applications are designed as sets of objects that interact and perform operations. Each object is a software module that models a physical or virtual entity and its related activity. Thus, an object exhibits state and behavior. Objects consist of data (sometimes called data members or properties), operations, and implementations (called methods). Operations are the interfaces to the object that initiate specific behaviors (actions or transformations). A method is the actual code that executes a requested operation.

As shown in Figure 3, most of the object's details, including its properties and methods, are hidden from application developers who use these objects. The process that isolates the internal workings of the object is called encapsulation.

One can think of an object as a self-contained specialist performing its tasks. It has its own data and the ability to perform certain actions on that data. To use an object, one sends it a message to perform a particular operation (function). This causes an operation to be invoked. The operation performs the appropriate method and optionally returns a response. The object can, in turn, send messages to other objects, requesting that they perform specialized services using their own data. A message contains the name of the object, the name of the operation, and the signature for the operation. A signature consists of the number, data type, and order of any input and output parameters required for the operation.

Figure 4 shows two objects communicating through the use of messages. The Flight Inquiry object sends a message requesting the flight schedule for United Flight 123. The message contains the name of the object providing the service (United123), the operation (GetFlightTime), and the signature ("Chicago" plus the return data type of the flight time).

In a distributed system, objects interact in the same way, at least from the object's perspective. In fact, the mechanisms for sending and receiving messages in a distributed system differ from those used within a single code space or system. In a distributed system, a message must be routed across a communication network into the correct system, and that system must route the message to the appropriate object. This routing of the message must remain transparent to the sending and the receiving object. Thus, client and server objects can reside on local or remote systems. (Maintaining this transparency is one function of an object request broker, a common service in a distributed client/server environment. When an object requests a service, the object request broker directs the message to the appropriate object to perform that service.)

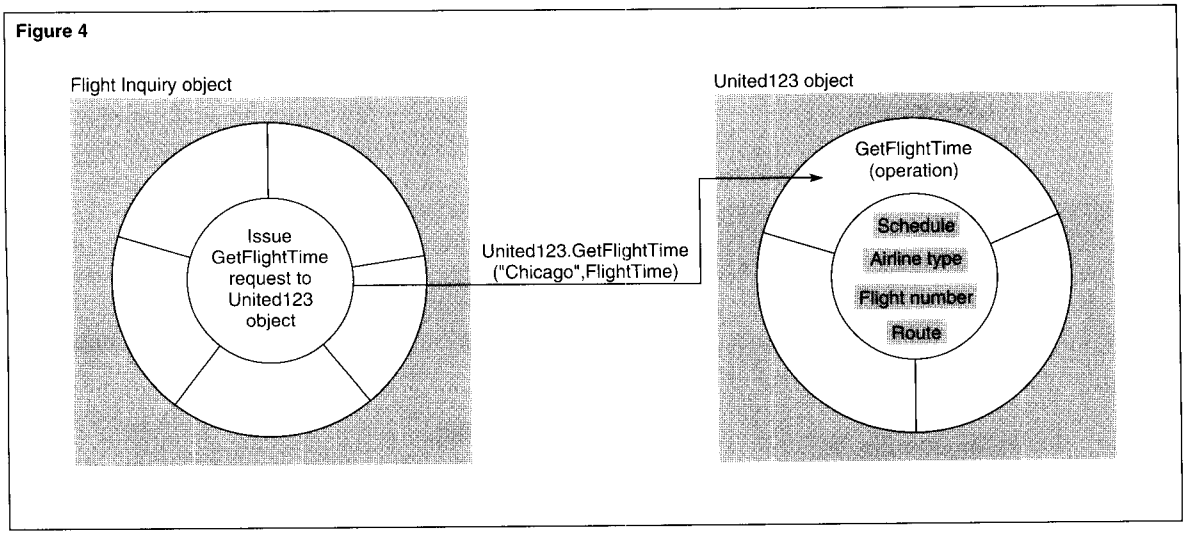


Figure 4.
Objects communicate by sending messages.

Establishing a standard architecture for distributed processing is crucial to realizing the benefits of the object approach. A practical architecture uses an abstract API as its interface and application middleware to implement the API. One can use implementations such as the RSC product, the Tandem Pathway Open Environment Toolkit (POET) software, the DCE RPC standard, or the TUXEDO TP monitor. Standard APIs are still emerging. For example, the industry-funded Object Management Group (OMG), of which Tandem is a member, has developed the Common Object Request Broker Architecture (CORBA).

Thus, one can envision an object-based system as a community of specialists cooperating to perform many complex functions, in which each specialist is responsible for maintaining its own information. (This concept resembles modern thinking about human organizations.) If this division of labor is accurate and communication among the specialists is good, the community can accomplish complex tasks quickly and efficiently.

Classes

Object technology simulates the way things work in the real world. To better understand object technology, one can examine its characteristics from a real-world perspective. Assume, for example, that one owns a factory that builds chairs and tables. To automate processes and procedures for the factory, one might define a Chair object and a Table object.

However, the designers expect to build more than one Chair or Table object. To describe more than one object having similar attributes and behavior, one uses a class. A class is a generic definition of a set of similar objects. It consists of the definitions of an object's data structure and the operations permitted on that data. Each instance of a given class is an object. Though two objects may appear to be the same (have the same properties and behavior), each will be identified separately.

Chairs and Tables have many attributes and behaviors in common. They both use wood, screws, and nails. They share other attributes such as size, cost, and time to build. One can extract these common attributes and behaviors to create a new, more general class. This process, called *abstraction*, is an important aspect of object technology.

Abstraction removes certain distinctions, so one can see commonalities between objects. Without abstraction, one would only know that each object is different. With abstraction, one selectively omits the distinguishing features of the objects, allowing one to concentrate on the features they share.

Figure 5.
Furniture is the superclass of the subclasses Chairs and Tables.

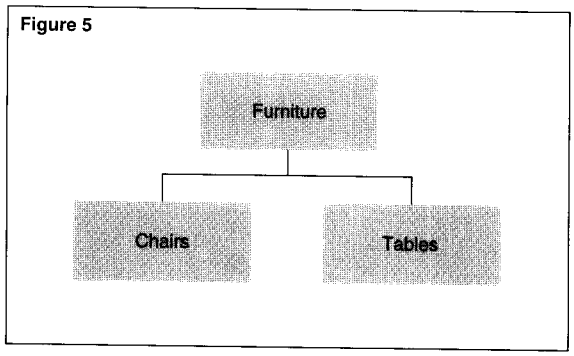
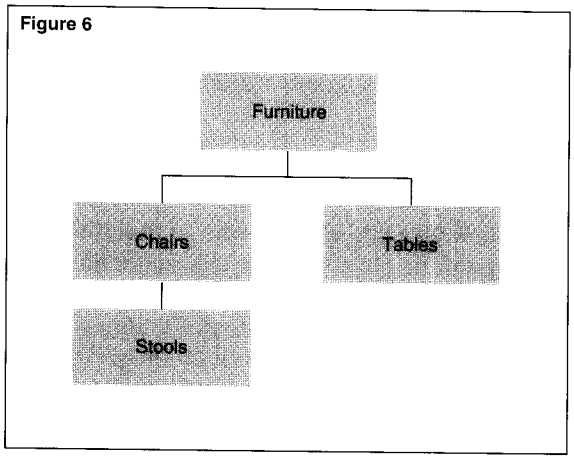


Figure 6.
Chairs is a subclass of Furniture and a superclass of Stools.



By using abstraction, one can formulate new classes of objects. This creates a hierarchy of classes that are generalized or specialized to each other. Classes higher up in a hierarchy are called *superclasses* to those at a lower level. Classes lower in the hierarchy are called *subclasses* to those at a higher level.

In this example, one might call the superclass Furniture. Since Furniture is a general name that includes Chairs, Tables, and perhaps other future items, one might want to make it an *abstract* class. An abstract class cannot be created. (No instances can exist.) It seems logical to make Furniture an abstract class, since it makes no sense to have a Furniture object.

Since one cannot create an abstract class, the only way to use it is to create derived classes, which inherit its attributes and behaviors. This type of derived class is called a *concrete* class because it can be instantiated. It is also a subclass of the abstract class. Figure 5 shows the relationship between the abstract or superclass (Furniture) and the subclasses (Chairs and Tables).

Inheritance

Both subclasses, Chairs and Tables, inherit all the attributes and behavior defined in Furniture. One implements inheritance by using an object-oriented programming language, which automatically maintains the relationships between superclass and subclass.

All Chairs and Tables could exhibit attributes such as size, cost, time to build, and material. However, attributes such as seat padding and chair backing are only important to Chairs. Thus, one would include these attributes in the Chair class but not the Table class.

Now assume the factory produces different types of chairs such as stools. Clearly, one should derive Stools from Chairs. Thus, as Figure 6 shows, Chairs would be the superclass and Stools would be the subclass of Chairs.

The Stools class inherits all the attributes and behavior contained in Chairs, which also includes what Chairs inherits from Furniture. Thus, inheritance greatly reduces the work of defining and coding Stools.

Stools, however, do not have backs. Though a subclass inherits attributes (such as chair backing) from its superclass, it can choose not to use them and override those attributes. (A subclass also inherits any operations and methods associated with the superclass. The C++ language supplies the mechanism for inheriting or overriding attributes, operations, and methods. The programmer controls this mechanism and can determine whether or not to use any given attribute or behavior inherited by a subclass.)

One can make a more precise model of the furniture hierarchy by extracting the unique attributes of Backed Chairs from the Chairs class. Using abstraction, one then revises the Chairs class. Figure 7 shows the new model, which contains two specializations of Chairs, NonBacked Chairs and Backed Chairs, and one specialization of NonBacked Chairs, Stools.

The example of a simple payroll system illustrates the principle that subclasses inherit behavior as well as properties. Assume the company in the example uses salaried and hourly employees. One might start with two classes, Salaried (employees) and Hourly (employees). By examining these classes, one can extract their common features. One can then create an abstract class, called Employee, which contains the commonality. (See Figure 8.) One makes Employee an abstract class, not a concrete one, because one expects all employees to have some type of pay specialization.

An important behavior common to both Salaried and Hourly classes is pay. Thus, one would elevate pay to the Employee class. By abstracting pay, one ensures that each type of employee does indeed get paid. This benefit occurs because all subclasses inherit the pay operation.

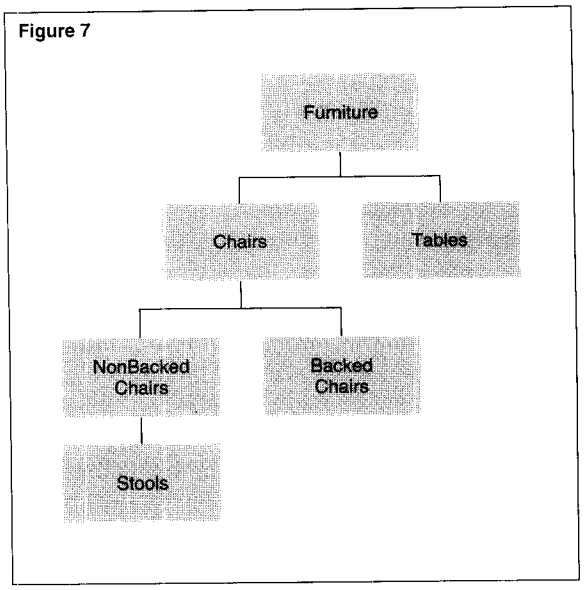


Figure 7.
Chairs becomes the superclass of Backed Chairs and NonBacked Chairs, which is a superclass of Stools.

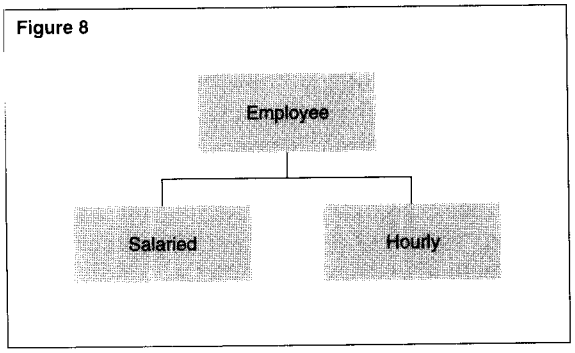


Figure 8.
The abstract class Employee is created from the Salaried and Hourly classes.

Polymorphism

Obviously, though, the method (implementation) of the pay operation must differ for different types of employees. Therefore, one places the operation and method in each subclass of Employee. One can thus use the same operation and signature with several different implementations. The application program can cycle through each employee, requesting that he or she be paid. In an object-oriented programming language, each employee is an object (an instance of a subclass of Employee). The requesting object sends a message to each Employee object, and the message will be executed by the correct object's operation.

This object-oriented principle is called polymorphism. Polymorphism is the ability of different objects to respond to the same message differently. Polymorphism enables one to define operations at a high level and let the details of the operation be handled by the object responsible for them. In this way, inheritance and polymorphism can greatly reduce the complexity of programming and testing code.

In a traditional program, one would write the pay function using some type of decision code to select the correct method of pay. For example:

```
...
If Employee-Type = "Salaried"
   then perform Salaried-Pay
Else
If Employee-Type = "Hourly"
   then perform Hourly-Pay
...
```

Compare this to the simple object-oriented statement, *EmployeeID.pay()*, in which the object-oriented language determines the selection. Here *EmployeeID* refers to an Employee object, and *pay* is the Employee object's operation. Through the use of polymorphism, the program automatically determines whether to use the Salaried object's pay or the Hourly object's pay.

Now assume that one must add a new type of employee, PartTime. In a traditional program, one would create the PartTime pay procedure and then modify the main program's decision code. Thus, one would modify code in at least two locations and perform a complete compile and link. In the object model, one simply creates PartTime as a subclass of Employee and binds it into the existing program. One does not have to make any changes in the existing code to correctly invoke the pay operation.

As these examples indicate, the inheritance relation between classes allows the definition and implementation of one class to be based on that of other existing classes. Inheritance is a most promising concept that can help developers realize the goal of constructing software systems from reusable components rather than hand-coding every system from scratch.

The inheritance relation is often called the *is-a* relation because subclasses have, by inheritance, all the features of their superclass. Thus, Salaried is-a(n) Employee. Undoubtedly Salaried has more features than Employee does, but whatever else it may be, it is also an Employee.

The is-a nature of inheritance is tightly coupled with the concept of polymorphism in a strongly typed object-oriented language. Because a subclass is-a superclass, the two classes can appear interchangeably in an object-oriented program. For example, whenever a program expects an instance of Salaried, Hourly, or PartTime, it allows one to reference them as an Employee.

Dynamic Binding

An object-oriented program implements polymorphism by means of dynamic binding. During compile time, the compiler cannot identify the operation that is called by the statement *EmployeeID.pay()*, since *EmployeeID* could refer to any one of its subclasses. Thus, the statement must be evaluated at run time, when it can tell what type of object *EmployeeID* refers to. This is known as dynamic or late binding. Thus, as the program executes the *EmployeeID.pay()* statement, it can determine the actual type of object and dynamically bind it to its appropriate pay operation.

It is this dynamic binding mechanism that allowed the subclass PartTime to be added (linked) to the employee program with minimum overall change and its pay operation to be called automatically even though it didn't exist in the original compiled application. This is very different from function calls in other languages such as C or COBOL. In these cases, the function is translated at compile time into a fixed address. This is called static or early binding and requires recompiling of the existing program.

The Benefits of Abstraction

The inheritance relation allows one to represent the structure of an application by identifying and encapsulating common functions in higher-level classes. One can then propagate these common functions to the subclasses that need them. Thus, inheritance allows developers to reuse a class that is almost, but not exactly, what they want, and to tailor the new class (or subclass) without introducing unwanted side effects into the original one.

Object technology supports this concept throughout the application life cycle. One can develop a software system using object-oriented analysis and design and implement it in an object-oriented programming language. Objects and classifications identified during analysis are preserved and enriched during design and directly implemented in code.

Furthermore, developers can accumulate the higher-level classes in a software repository. As the repository grows, developers will be able to find in it a generalization for almost any desired class. Many developers believe this ability to capture and encapsulate abstraction directly in code represents a major breakthrough in software technology.

An Improved Development Methodology

Most computer professionals know several programming languages and several diagramming techniques for representing design details. Yet most programmers know only one approach to system design. It is perhaps harder to learn a new system-development technique than it is to learn a new language. To learn a language, one memorizes a few keywords and their valid arrangements. To learn a new system-development technique, one must fundamentally change one's way of thinking.

The object-oriented design paradigm gives developers a way to model real-world problems by combining related procedures and data in a flexible, consistent structure (an object). The analysis and design stages of the application life cycle remain separate activities, but they are closely related. Developers use both stages to build a model of the business problem. Although there is no single standard notation for object modeling, there are many tools that support most popular notations (such as Rumbaugh, Booch, and Coad-Yourden).

Developers construct the model by viewing the business problem as a set of interacting objects and the relationships among them. They then assemble software-based models of these objects and their relationships to form the basic architecture of the application. The information developed in the analysis stage becomes an integral part of the design rather than simply providing input into the design stage. This smooth transition is facilitated by the homogeneity of the pieces being used by each process. This homogeneity contrasts starkly with the difference in point of view between structured analysis and structured design, which traditionally use different notation and semantics.

One does not necessarily discover these pieces in a top-down approach, as in procedural decomposition. Neither does one have to use a bottom-up approach, starting with low-level classes and abstracting to higher-level classes. The discovery can take place at any level and then be subclassed or superclassed as needed.

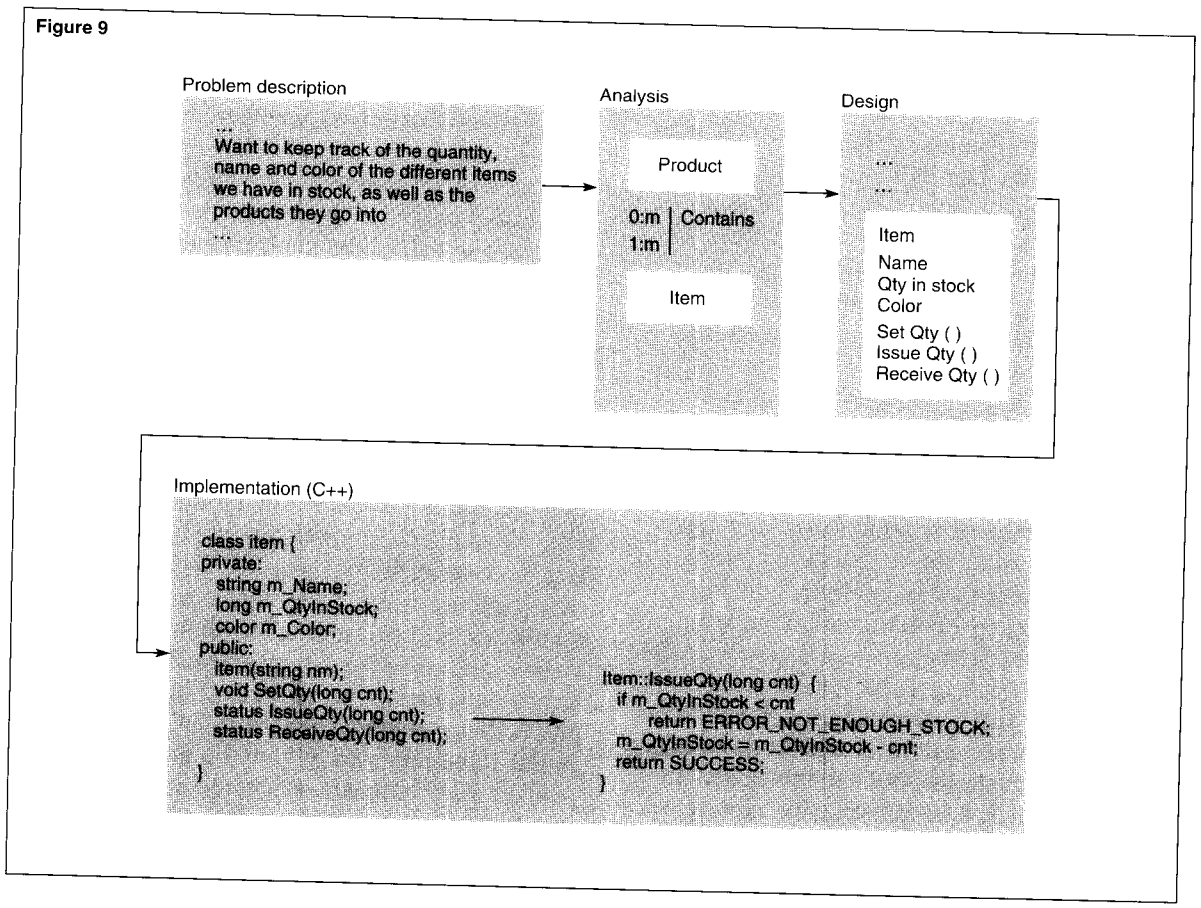


Figure 9.
Evolution of an application from a problem description to the finished code.

One can relate many objects, classes, and relationships directly to the original problem. Though many classes do not represent physical objects, they are conceptual entities that can be stated in the terminology of the business problem. The common notational model used throughout the life cycle is called the object model.

Figure 9 is a simple illustration showing the evolution of an application from a problem description to the finished code. It shows how the object model fosters a natural progression from analysis to design to implementation.

One uses the same conceptual model in object analysis and design; the design builds on the core object model created by analysis. The object paradigm encourages iteration, discovery, and interaction at all levels of the organization. It allows designers to focus on the business solution rather than the final implementation details. At the system design stage, one can determine the physical splitting and assignment of object locations, thus avoiding many of the incompatibilities caused by separating client and server development early in the life cycle. Another benefit of using the object paradigm is that one can allow an object, once defined, to flow through the life cycle as a stand-alone entity. This helps to break down a large project into small, manageable pieces.

Hiding and Encapsulating Implementations

As developers begin the system design and implementation stage of the life cycle, they need to address the issue of maintaining technology independence. One way to accomplish this is to encapsulate the underlying technology and hide its implementation from the rest of the application. The logical layers shown in Figure 10 can form precise lines, establishing barriers between the technologies used in each layer.

Figure 10 shows veneer layers, called API guards (or API isolation layers), that surround the business logic. Often called the user-interface guard, middleware guard, and DBMS guard, these API guard layers protect the business logic from each implementation layer. An API guard layer consists of an abstract API that is used by the business logic and invokes the actual implementation technology. Mechanisms in the API guard layer map the abstract API to a meaningful implementation of the API.

Assume, for example, that the business logic uses an application-middleware guard layer to issue messages to remote objects. The guard layer needs to use methods that can write requests (issue messages) and read replies or messages issued to it from other objects. An abstract API could consist of only two verbs, READ and WRITE. The guard layer would logically map these verbs into the correct implementation of the API, which would establish a session with a remote host, write a message, read a reply or message, and close the session. The guard layer would hide the transformation mechanisms and implementation from the application developer. All business applications would use this API as their interface.

To change technology, one would rewrite the middleware guard layer once instead of rewriting every client. This alone would be a benefit, but if one only wanted to protect the business logic from a single technology, one could do it without using object technology. Guard layers, however, should be able to protect the business application from multiple technologies that may be added or changed in the future.

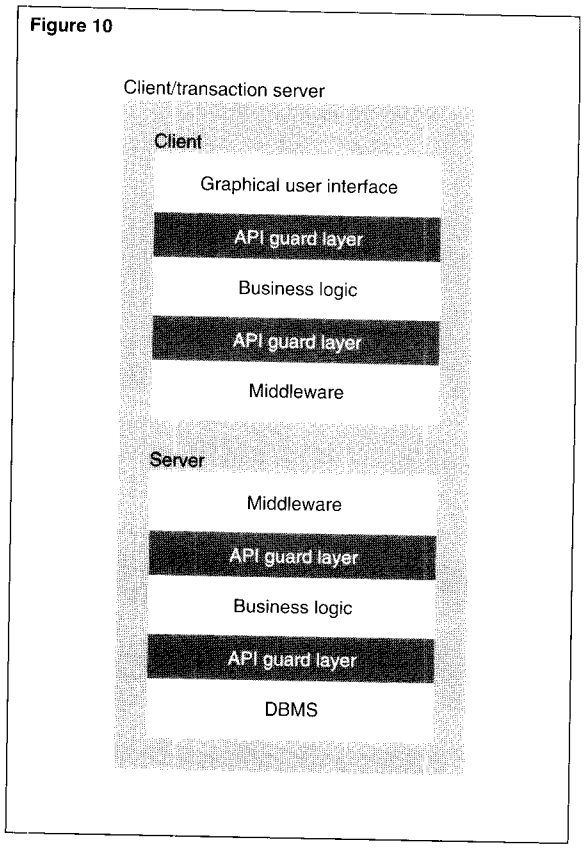


Figure 10.
Encapsulating functional layers by using API guard layers.

Assume, for example, that an application uses POET as the middleware. Now one wants, in addition, to communicate with a process developed for access by a DCE RPC request (on a Tandem server or another host). One may also want to access data on the local LAN server using a Windows NT RPC call. (This example mixes and matches application middleware components, illustrating a possible requirement of enterprise client/server computing.)

Figure 11.
Without an API guard layer, each RPC and RSC request must be hard-coded.

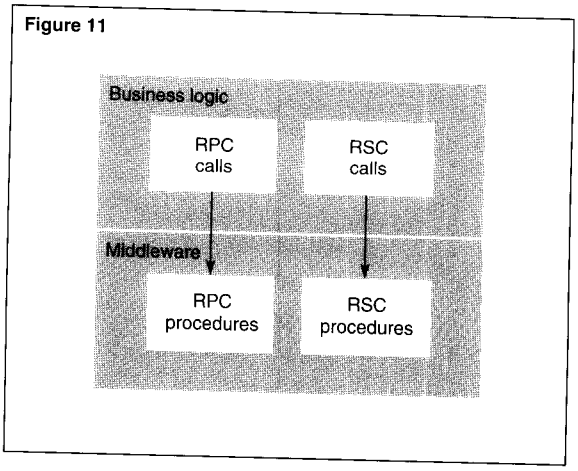
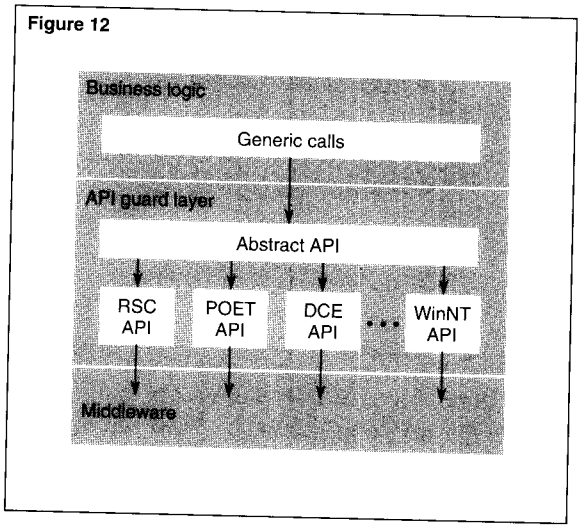


Figure 12.
Adding an API guard layer increases the interoperability of the requested services.



In a client/server model that uses fuzzy boundaries, adding these middleware components would require a complete rewrite of the application. Also, all the developers would have to learn the new application middleware technology. Even a well-defined implementation without an API guard layer would require that developers intertwine the implementation-specific code with the business logic. As shown in Figure 11, developers would have to hard-code each different request into the application. Thus, boundaries would become fuzzy, even if they were well defined before the addition of the new middleware.

If developers had created API guard layers, it would be much easier to perform this task. Only a small group of developers would need to learn the new technology. They would incorporate the new functions in the middleware guard layers. The new calls would remain transparent to the developers responsible for the business logic, who would see the same interfaces to the API guard layers they'd seen before. As shown in Figure 12, API guard layers add transparency and increase the interoperability of the requested services. Thus, they help to prevent future rewrites of the business logic.

Object technology simplifies the task of protecting business logic. It allows the implementations to be encapsulated into subclasses of the existing guard layer. Inheritance, polymorphism, dynamic binding, and other object-oriented techniques all help to reduce the amount of code one must write, test, and place into production.

Guard layers are usually written once by the systems programmers and then used throughout the enterprise. The guard layers stabilize the business logic by encapsulating the uniqueness of each layer. When one migrates to new implementation-specific technologies, the guard layers prevent the business layers from being corrupted. They also allow one to interchange technologies with little impact on the application.

Creating Reusable Building Blocks

After developers implement the general application layers, they can begin implementing (coding) the specific objects that make up the business application. At this stage, the object model provides design techniques and language features to support the creation of reusable common services and components.

The reuse comes in a variety of forms. Some reuse in the object-oriented paradigm is the same as that in the procedural paradigm. However, in the object model, each time one creates an instance of a class, reuse occurs. This is similar to declaring a variable of a specific type. The difference is that an instance of a class is a more complex structure than a simple variable. An instance of a class combines data structures and operators on those data structures.

Inheritance supports reuse at two levels: during high-level design and low-level design. During high-level design, inheritance allows one to model generalization-specialization relationships, which appear in the form of classifications. One can view a Binary File as a special type (subclass) of File as well as a general description (superclass) of a more specific class such as an Image Binary File, Fax Binary File, or Object Binary File. The high-level use of inheritance encourages the development of useful abstractions, which encourages reuse.

In practice, designers often recognize similar midlevel abstractions separately. For example, they may create a Binary File and an Edit File. Inheritance allows them to identify common elements among abstractions and produce higher-level abstraction, such as File, from those common elements. This commonality then becomes available to be reused later in the current design or in future designs.

For example, designers may later identify Repository Files and Sound Binary Files. The existing File abstraction may provide a large part of the description of these classes, including attributes such as creation date, size, modified, and bit-storage formation. The benefits of reuse prompt designers to search for higher and higher levels of abstraction.

Inheritance in Low-Level Design

During low-level design, inheritance allows developers to use an existing class (in the form of bindable object code) as the basis for defining a new class. Inheritance alleviates the problems associated with morphallaxis by making the new class dependent on the existing one.

This form of reuse is nonintrusive because the existing code is not modified. Thus, the new code in the new class cannot cause the existing code to break. When the new class definition is compiled, the inherited code is automatically included. Any modifications in the original class, such as bug fixes or additional features, are incorporated into the new class at the next compilation. This technique ensures that all instances of inherited code remain consistent. It allows a class to serve as the basis of many new definitions without propagating the errors of the original definitions throughout the system.

Often the functionality of a class makes sense conceptually for other kinds of data types. At the programming level, a source-code construct in C++ that promotes this reusability is called the *parameterized type* or *class template*. A class template acts as a skeleton used by the compiler to create a new class, using a specific data type, at compilation time. For example, consider building a class that handles lists. The required operations do not depend on the type of objects in the list. Instead of rewriting a list class for every new kind of object, one can write a single class template and reuse it as needed. One can thus reduce future programming efforts. The ability to create libraries of generic class templates could prove to be a central concept in the future of software development.

Creating Reusable Frameworks

By using object-oriented principles, application developers can create client/server frameworks consisting of reusable components to bridge the gap in technology between workstations and hosts. Only a few highly skilled developers need to produce the frameworks. Most developers do not have to know the technologies that the frameworks encapsulate; they can concentrate on developing the business application. Thus, frameworks offer a fast start for development groups.

A client/server framework of compatible software components allows programmers to build reliable, open applications. Drawing on reusable components, programmers can construct most of these applications instead of coding them. Programmers do not need to be concerned with the details of how services get implemented, but rather with their connections and high-level functions. These services may reside in a single client or server, or they may be implemented across multiple client and server platforms (not only workstations and host machines).

A client/server framework provides a variety of required service components that are packaged into software like integrated circuits (ICs). As with hardware ICs, one does not have to understand the complicated logic inside them to use their functions. Instead, one only needs to understand their user interfaces.

An open framework allows one to plug in software ICs wherever they are required by applications. One can interchange software ICs with similar ones that support different technology. The internal implementation of a function may be different, but the user interfaces remain the same. Assume, for example, that a user wants to employ the Windows-based CPI-C software as the transport mechanism instead of RSC. To do this, the programmer replaces the RSC chip with the WinCPIC chip, without having to change the application.

One creates reusable frameworks and components by using an object-oriented programming language such as Tandem's C++ Translator product (an implementation of CFront) and a C++ language on the workstation. The resulting object code, residing in class libraries, can then be bound into the applications as needed. On the Tandem system, one can use the Common Run-Time Environment (CRE) to bind the components into COBOL, TAL, C, and C++ application object code. On the workstation, one can bind the components into application object code, use them in dynamic link libraries, or use them as stand-alone daemon processes.

One could achieve some of the benefits described in this article in traditional development environments. Those environments, however, require a strict and rigorous structure as well as external control. Object technology provides an environment that more naturally produces these benefits.

Conclusion

The client/server model offers great benefits but has an imprecise architecture. In particular, three issues can affect the development of flexible, durable client/server applications: dependence on a particular technology, inconsistent development methodology, and lack of reusable software.

Object technology can extend the client/server model in a way that alleviates many of these problems. Concepts such as inheritance, abstraction, and polymorphism allow one to model real-world processes and implement that model directly in code. These concepts and techniques support a consistent development methodology that promotes reusability.

By using object frameworks and components that support encapsulation, object technology also reduces the problems of technology dependence and morphallaxis. In addition, one can use frameworks, together with libraries of high-level classes, as building blocks that make future applications easier to create.

Bibliography

Booch, G. 1991. *Object-Oriented Design with Applications*. Addison-Wesley.

Christerson, M. et al. 1992. *Object Oriented Software Engineering: A Use Case Driven Approach*. Addison-Wesley.

Coad, P. 1993. *The Object Game*. Object International.

Coad, P. and Nicola, J. 1993. *Object-Oriented Programming*. Prentice-Hall.

Coplien, J. 1992. *Advanced C++: Programming Styles and Idioms*. Addison-Wesley.

Eckel, B. 1993. *C++ Inside and Out*. McGraw-Hill.

Lippman, S. 1991. *C++ Primer*. Addison-Wesley.

Martin, J. and Odell, J. 1992. *Object Oriented Analysis and Design*. Prentice-Hall.

Meyer, B. 1992. *Object-Oriented Software Construction*. Prentice-Hall.

Meyers, S. 1992. *Effective C++: 50 Specific Ways to Improve Your Program Design*. Addison-Wesley.

Object Management Group. 1993. *Common Object Request Broker (CORBA)*. QED.

Rumbaugh, J. et al. 1991. *Object-Oriented Modeling and Design*. Prentice-Hall.

Acknowledgments

I would like to thank the reviewers who commented on and helped to improve this article.

Tom Rohner is a consulting analyst working in Technical Marketing Services (TMS). Tom has specialized in client/server computing since 1988 and object technology since 1990. During that time Tom helped bring the RSC product to market, developed and delivered the initial Client/Server Training course for Tandem analysts, wrote the Client/Server Construction professional service, and has helped many Tandem users architect and implement client/server solutions.

Basic Uses and New Features of Extended GDS

The Tandem™ Extended General Device Support (GDSX) product[1] is designed to simplify the development of front-end and back-end processes for communication with I/O devices. The I/O devices can be of any type, including workstations, terminals, ATMs, point of sale (POS) devices, and industrial robots. In a GDSX front-end process, input from multiple I/O devices is processed in separate tasks and forwarded to processes on a Tandem host. This is illustrated in Figure 1. Common uses of GDSX front-end processes include data-stream conversion between formats used by I/O devices and formats supported on a Tandem system, implementation of communication protocols, and error handling for errors in network communication.

[1]GDSX refers to versions of the General Device Support (GDS) product starting with release C30 of the NonStop™ Kernel operating system. At that time, GDS was modified to allow I/O from extended memory. This greatly expanded GDS limits on I/O handling and multitasking.

A GDSX back-end process receives input from multiple processes on a Tandem host and provides access to a limited number of I/O devices. Common uses of GDSX back-end processes include implementation of communication protocols; message switching; and coordination of access to shared resources or I/O devices, such as ports on a remote system, log files, or terminals. Figure 2 illustrates a GDSX back-end process that provides message switching for host processes and implements a communication protocol for sending messages to a remote system.

Typically, developing a front-end or back-end process that interacts with major Tandem software components and can handle large numbers of I/O devices or host processes requires a significant development effort and complex, multithreaded coding. GDSX software provides a multitasking environment that allows developers to write simpler, single-threaded code and achieve multithreaded results. In addition, GDSX software supports fault-tolerant processing and provides interfaces to the Tandem Subsystem Control Facility (SCF), Event Management Service (EMS), and Subsystem Programmatic Interface (SPI) software products.

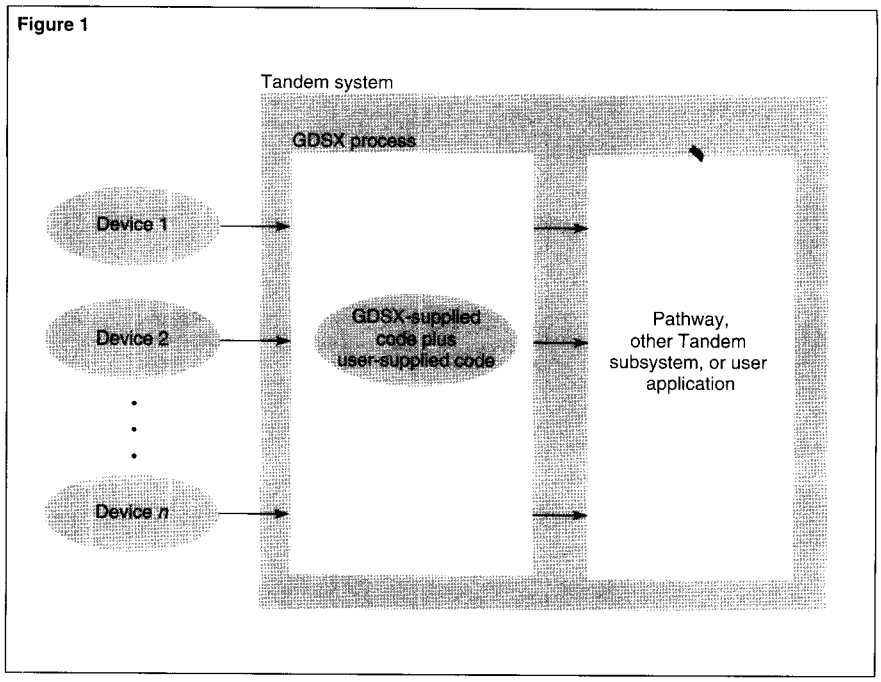
GDSX-supplied code for D-series releases of the Tandem NonStop™ Kernel operating system adds features not available under C-series releases: access to server classes through the Pathsend facility in Pathway, TMF™ (Transaction Monitoring Facility) support for GDSX processes, and enhanced messaging between tasks within a GDSX process. Accessing servers through the Pathsend facility rather than through Pathway requesters can significantly improve application response times. TMF support for GDSX processes means that TMF protection for audited tables or files can begin as soon as a GDSX process receives input from an I/O device and can continue up to the point at which the GDSX process is ready to send a reply back to the device. GDSX intertask messaging makes it possible to develop special-purpose tasks that increase the efficiency of a GDSX process.

This article describes uses of the GDSX product and enhancements made available with D-series releases. Readers should be generally familiar with the Pathway transaction processing system, the Pathsend facility, and the Subsystem Control Facility product.

User-Coded Device Handlers and Line Handlers in GDSX Processes

A GDSX process, front-end or back-end, combines user-written code (user code) with GDSX-supplied code. The user-coded portions of a GDSX process can be written in either the TAL or C programming language.[2] Two types of user-written components are of particular importance, *device handlers* and *line handlers*. Every GDSX process requires a user-written device handler; some, but not all, back-end processes also use a line handler. Figure 3 shows device-handler threads in the generalized GDSX front-end process of Figure 1. Figure 4 shows device-handler threads and a line-handler thread in the GDSX back-end process of Figure 2.

[2]GDSX provides data structures and other types of declarations for user code written in the TAL programming language. Users must write their own declarations to serve as an interface between GDSX-supplied code and user code written in C.



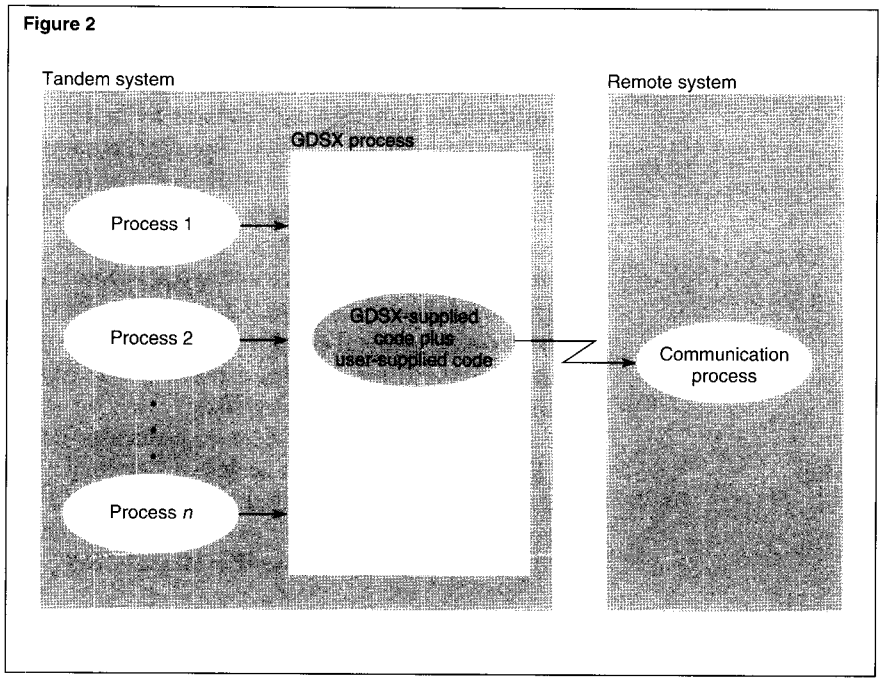


Figure 1. *GDSX front-end process.*

Figure 2. *GDSX back-end process.*

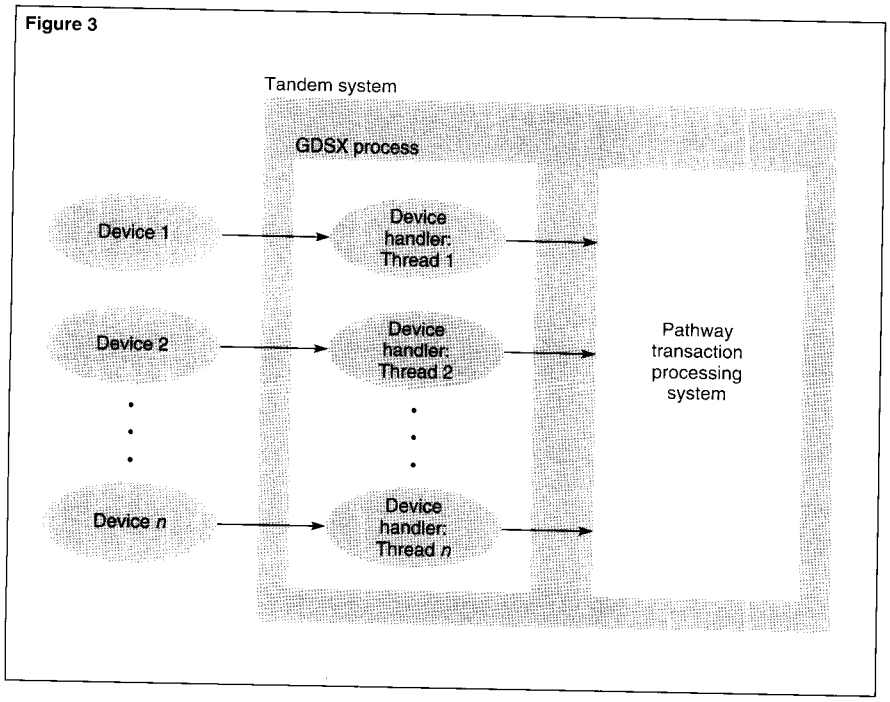


Figure 3.
Device-handler threads in a GDSX front-end process.

As shown in Figure 3, in a GDSX front-end process, individual I/O devices are associated with specific device-handler threads. GDSX multitasking generates the threads from the user-written device handler. The device handler can be coded to provide a wide range of functions, including data-stream conversion, implementation of communication protocols, and error handling for errors in network communication. As described later, a device handler can also execute special-purpose tasks such as validating user authorizations, accessing security boxes, and maintaining log files.

In a GDSX back-end process, processes on a Tandem host send data to device-handler threads and, in contrast to a GDSX front-end process, there is no direct association between I/O devices and device-handler threads. As stated earlier, common uses of a GDSX back-end process include implementation of nonstandard communication protocols, message switching, and coordination of access to shared resources or I/O devices, such as log files, terminals, and remote ports.

In Figure 4, processes on the Tandem host need to send data to a communication process on a remote system. Each process sends its data to a specific device-handler thread. The device-handler thread, possibly after some manipulation of the data, forwards the data to a line-handler thread. GDSX generates the line-handler thread from a user-coded line handler that implements the communication protocol required on the remote system. GDSX-supplied code provides queuing for device-handler threads so that data from one thread does not interfere with the data from another.

GDSX Multitasking Environment for Front-End Processes

In a GDSX front-end process, there is an explicit association between I/O devices and individual device-handler threads. In Figure 3, among many other possibilities, the device-handler threads can be thought of as implementing a nonstandard protocol for communication with remote workstations, ATMs, or POS devices, or as performing data-stream conversion for industrial robots, nonstandard terminals, ATMs, or POS devices.

In a GDSX front-end process, device-handler threads are created by defining subdevices through the GDSX-SCF interface. First, a device-handler task is defined by adding it as a subdevice (ADD SU). It can then be associated with a physical I/O device and started with the SCF START SU command. GDSX acts like its own operating system in respect to device-handler tasks. It maintains one copy of the device-handler object code in memory. When the subdevice for a device-handler task is started, GDSX-supplied code creates entries

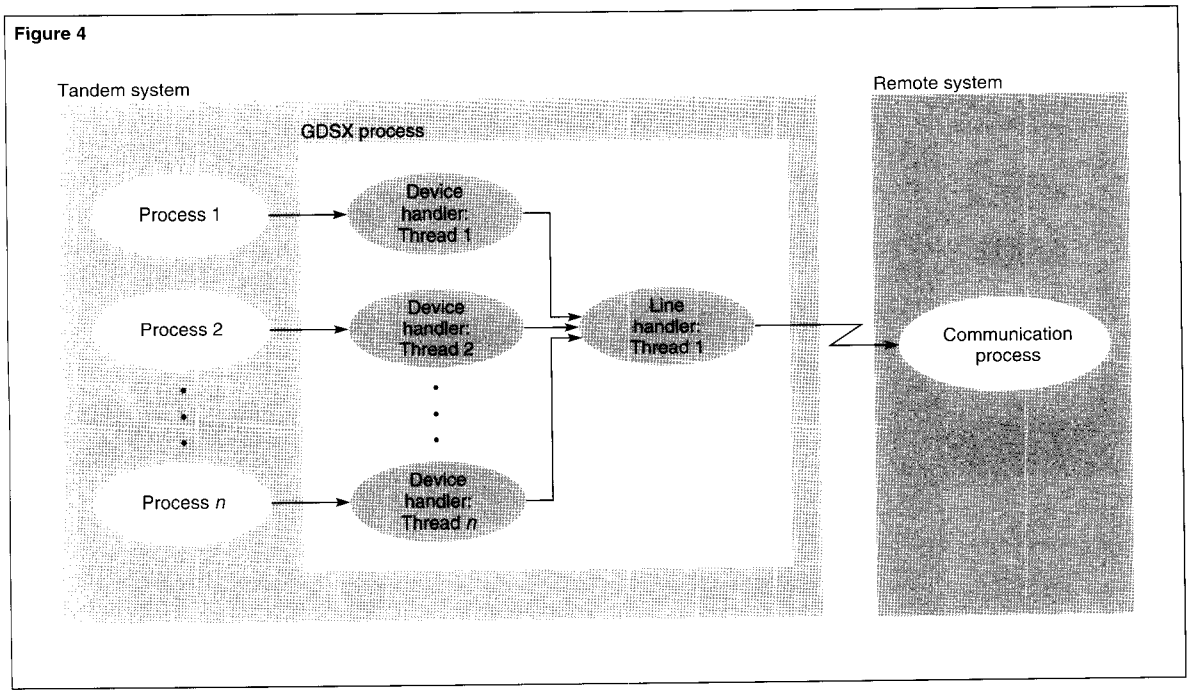


Figure 4.
Device-handler threads and a line-handler thread in a GDSX back-end process.

for the task in its own internal tables and starts executing the device-handler object code on behalf of the task. This is now the active device-handler task. Only one task can be active in a GDSX process at a time. The currently active task executes until it needs to wait for a resource or for the completion of an I/O. At this point, it can be suspended and swapped to extended memory and another task made active. The newly active task can be brought in from extended memory, if it had been active earlier and was swapped to extended memory, or it can come from a freshly started GDSX subdevice.

When a task is swapped to extended memory, the device-handler code is not touched. GDSX maps data for the task to extended memory and uses its own tables to keep track of the data location, the point at which the task stopped executing instructions in the device-handler code, and additional housekeeping information about the task. When GDSX software activates a task saved in extended memory, it moves data for the task back into active memory and begins executing device-handler code from the point at which it had stopped when the task was previously active.

GDSX Multitasking for Back-End Processes

In a GDSX back-end process, Tandem host processes need to communicate with a limited number of I/O devices. Data from the Tandem processes is directed to individual device-handler threads. In many cases, the device-handler threads process the data and forward it to the I/O devices without using a line handler. A user-written line handler is useful when the data needs to be sent across a communication line. In this case, device-handler threads send their data to a line-handler thread, which typically implements a communication protocol and forwards the data to a remote system. GDSX-supplied code queues the data sent to a line-handler thread in order to prevent data from one device-handler thread from interfering with data from another.

Figure 5.
GDSX back-end process with two line-handler threads.

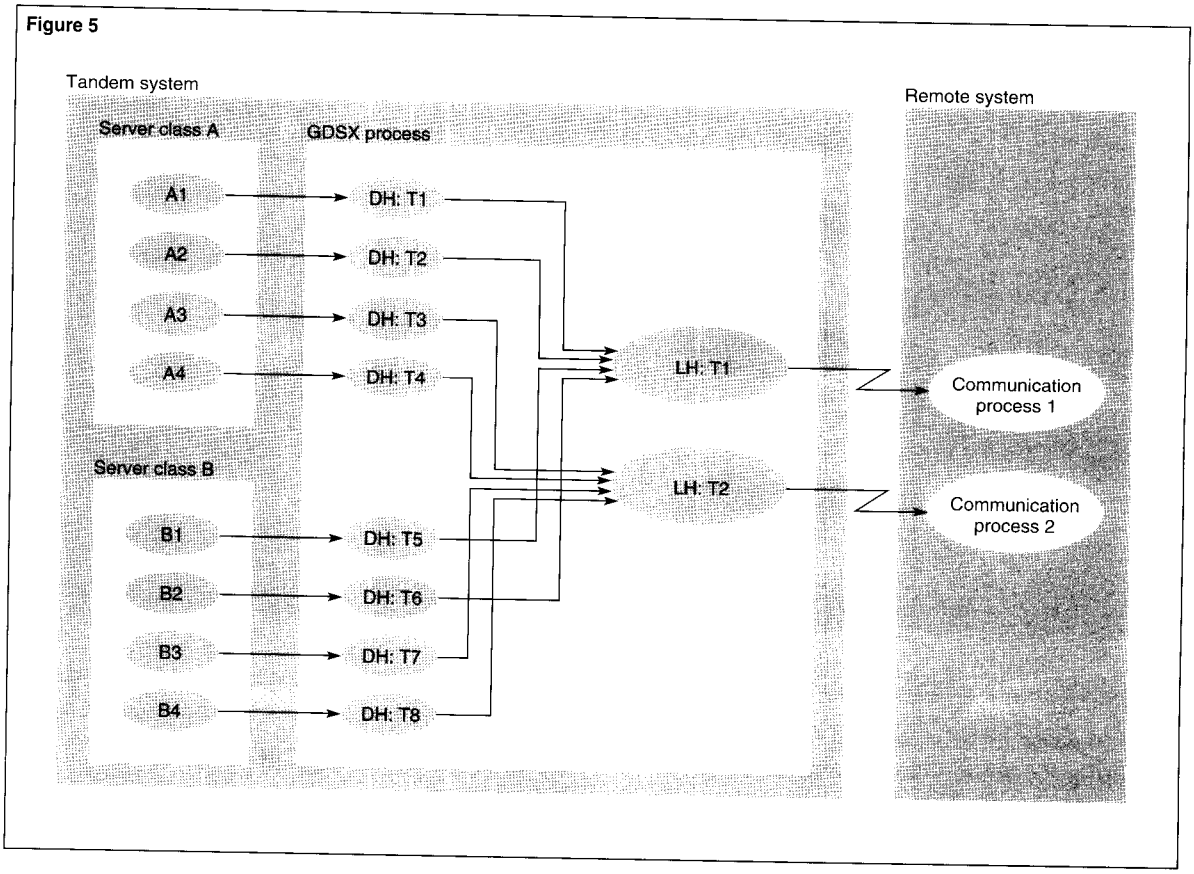


Figure 4 illustrates a simple back-end process in which servers communicate with a remote system through device-handler threads and a single line-handler thread. There can also be multiple line-handler threads in a GDSX back-end process. When a line handler is used, device-handler threads are configured under line-handler threads, so that a device-handler thread always sends its data to the same line-handler thread. Figure 5 shows a GDSX back-end process with two line-handler threads. There are two server classes, each with four servers. The servers send data to GDSX device-handler threads. The device-handler threads are configured so that each line-handler thread receives data from two servers in server class A and two servers from server class B. This configuration provides protection in the event that either one of the communication lines fails.

In a GDSX back-end process with a line handler, individual line-handler threads are created by defining and starting lines through the GDSX-SCF interface. When a GDSX line is started (START LINE), GDSX-supplied code creates entries for a line-handler thread in its own internal tables and starts executing line-handler object code on behalf of the thread. Device-handler threads that send data to a given line-handler thread are defined as subdevices on the line corresponding to the thread. Creating line-handler threads and assigning device-handler threads to them is described in detail in the *Extended General Device Support (GDSX) Manual* (1993).

A user-coded line handler can simplify the development of a GDSX back-end process for sending data to a remote system over a communication line. More generally, a line handler is advised for any GDSX process that needs to manage synchronous I/O to a device. In most other cases, a GDSX process without a line handler is preferable.

In combination with GDSX-supplied code, a user-coded device handler can always achieve the same results as a line handler. GDSX D-series releases make this feasible primarily through the implementation of GDSX intertask communication and the use of special-purpose device-handler tasks, which are described in the next section.

Figure 6

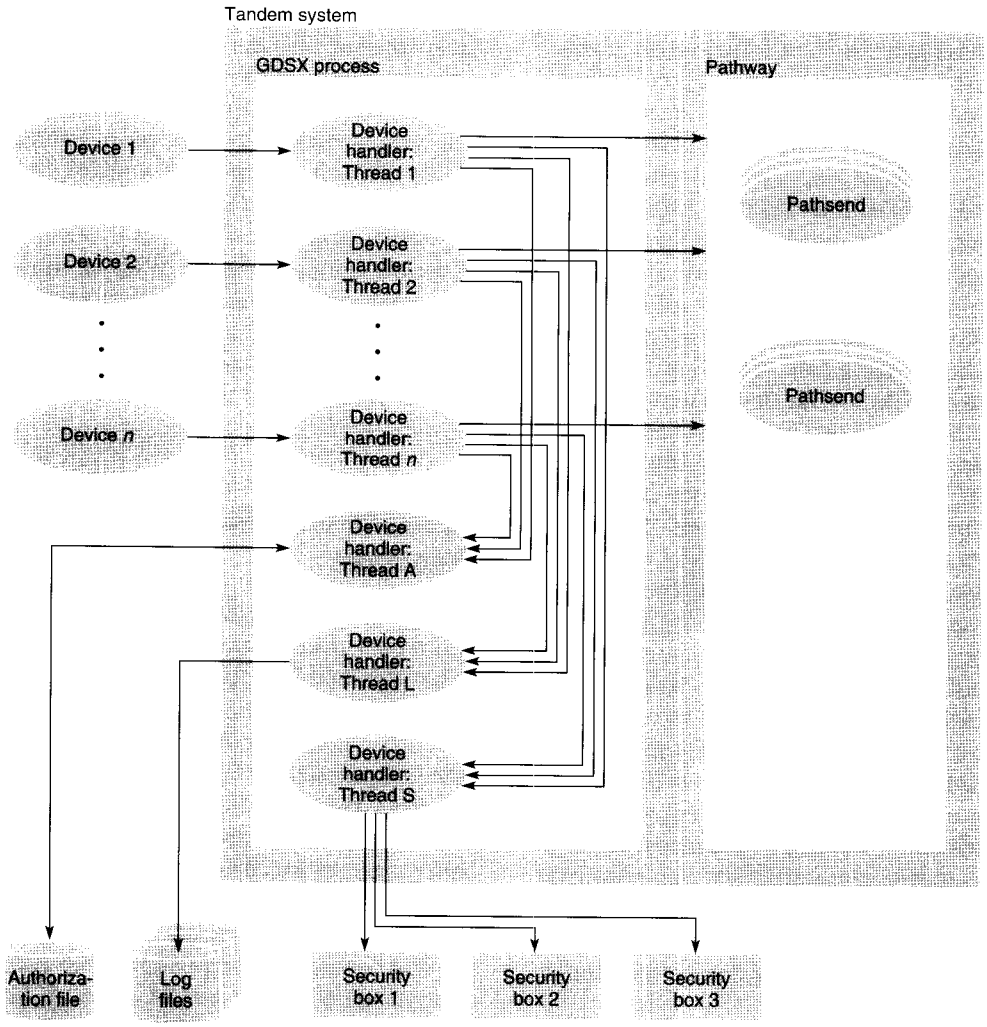


Figure 6.
GDSX intertask communication and special-purpose tasks.

GDSX Intertask Communication and Special-Purpose Device-Handler Tasks

GDSX-supplied code provides intertask communication for threads in a GDSX process without going through the NonStop Kernel message system. GDSX also provides its own queuing for intertask messages sent to the same thread. These features make communication between GDSX threads highly efficient and favor the development of special-purpose device-handler tasks for providing services to front-end device-handler tasks associated with I/O devices. The specialized tasks carry out operations that would otherwise have to be executed separately within each front-end task. Special-purpose device-handler tasks can also be implemented in GDSX back-end processes.

Figure 6 illustrates three special-purpose device-handler tasks that rely on GDSX intertask communication. Each special-purpose task represents a separate procedure coded in the device handler that provides data-stream conversion or other functions for threads 1 through *n*.

Thread A in Figure 6 is an authorization task that verifies authorizations for each device-handler thread associated with a physical I/O device. By having a single task handle all authorizations, only one OPEN on the authorization file is necessary and no overhead is incurred for checking file locks and file accesses by other tasks. Without a specialized authorization task, device-handler threads 1 through *n* would have to individually check for file locks and file accesses and carry out any application logic required for evaluating authorizations.

The second special-purpose task in Figure 6, thread L, is a logging task. The logging task writes to a log file. Under designated circumstances, it opens a new log file and closes the old log file. For example, the logging task may monitor the time and check for user commands and error messages. Using the time, it can switch to a new log file every day at midnight. In response to a user command or file-system error message, it can open a new log file. Typically, a logging task of this type also makes sure that all database information belonging to the same transaction is written to the same log file, even if the transaction starts with one log file and ends when a new log file is in use.

Having a special-purpose logging task monitor the time and check for event messages and error messages is more efficient than requiring every task associated with an I/O device to do this on its own. Further, in the absence of a special purpose task, individual device-handler threads have to compete with each other for access to the log file. With a large number of device handler threads, this contention can slow performance. The contention is eliminated when a special-purpose logging task is used.

The third special-purpose task in Figure 6 is a security task. This task monitors the availability of each security box, receives messages from device-handler threads, queues them if necessary, and forwards them to security boxes as they become available. Without the security task, each device-handler thread associated with an I/O device would have to look for a free security box on its own and then contend with other threads for access to it.

Pathsend and TMF Support for GDSX Processes

Prior to D-series releases, GDSX processes could only access servers through Pathway requesters and TMF protection could only be applied to transactions and database activities executed outside a GDSX process. This situation is illustrated in Figure 7.

GDSX D-series releases make it possible to build requester functions into a GDSX process and access servers directly through the Pathsend facility in Pathway. This can significantly improve application response times. In addition, GDSX D-series releases extend TMF protection to cover GDSX processes. As a result, protection of audited database files can now begin in a GDSX process as soon as the process receives input from an I/O device and can continue up to the point at which the process sends a reply back to the device. Figure 8 illustrates a GDSX process that contains the functionality of a requester and accesses servers through the Pathsend facility.

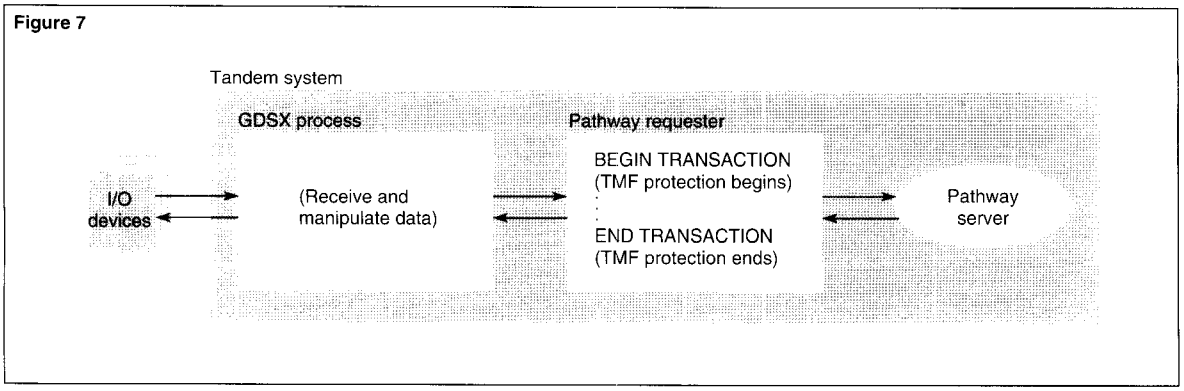


Figure 7.
A GDSX process using a Pathway requester to access a server.

Improved Performance Through Pathsend Servers

Directly accessing Pathsend servers from a GDSX process, rather than going through Pathway requesters, improves performance by avoiding use of a terminal control process (TCP) and by using compiled code instead of interpreted code. Multithreaded TCPs regulate access to Pathway requesters. Bypassing the use of a TCP reduces messaging costs and possible queuing delays in reaching a server. Compiled code is executed much faster than interpreted code. Pathway requesters are written in the SCREEN COBOL language and are interpreted. GDSX device handlers can be written in the TAL or C programming language, and are compiled. In combination, these factors can have a considerable effect on performance. At one GDSX site, moving from Pathway SCREEN COBOL requesters (as in Figure 7) to compiled requester functions within a GDSX process (as in Figure 8) improved overall application performance by 50 percent.

If a GDSX process bypasses Pathway SCREEN COBOL requesters and uses the Pathsend facility, the user-written device handler in the process must provide functions that would have been in the Pathway requester. The additional coding is usually very simple. GDSX front-end processes typically serve intelligent I/O devices that provide their own screen displays. SCREEN COBOL requesters for such devices primarily define and control Pathway intelligent device support (IDS) messages and forward data to servers. To replace a SCREEN COBOL requester, a device handler that provides data-stream conversion or other functions may only need added procedure calls for using the Pathsend facility and code for accessing the appropriate server class.

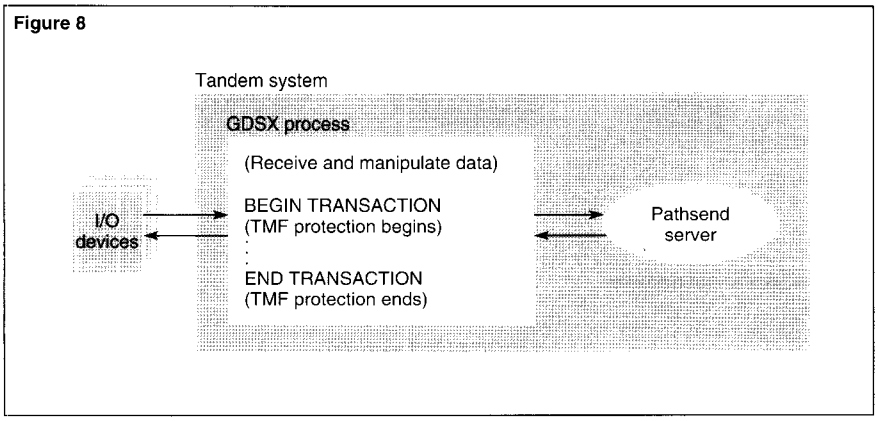
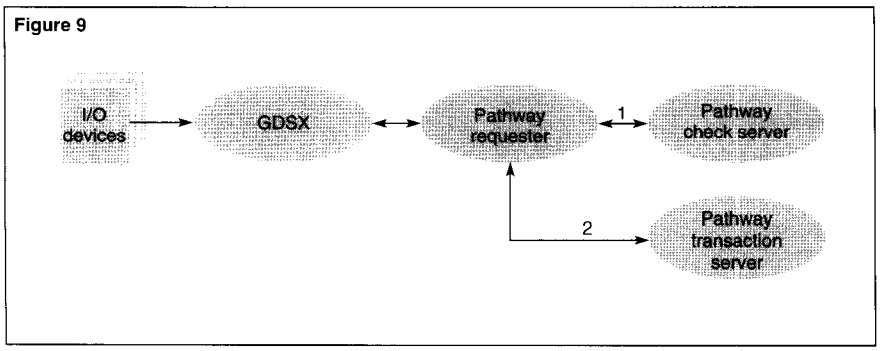


Figure 8.
A GDSX process accessing a server through the Pathsend facility.

Keeping Server Logic Outside of GDSX

Just as the functions of a requester can be placed in a device handler, it is also possible to put server logic in a device handler. Combining both requester and server functions in a single GDSX process would make it unnecessary to use either Pathway TCPs or the Pathsend facility to access servers. However, in most cases, this approach is not recommended. If both requester and server logic is carried out within the same GDSX process, one CPU must do requester-server processing for all the threads in the GDSX process. As a result, while the CPU is executing server logic for one thread, all other threads and requester functions are kept waiting.



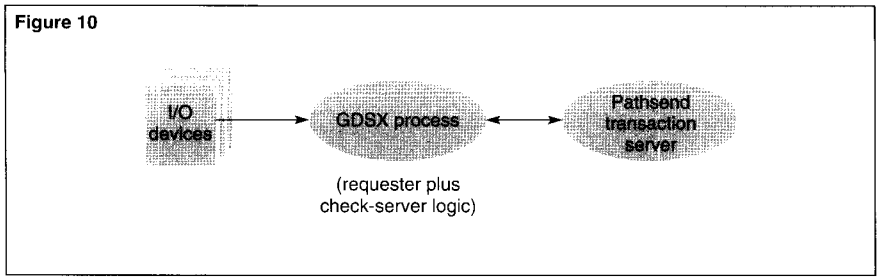


Figure 9.
Configuration with Pathway requester and check server.

Figure 10.
Configuration with requester and check-server logic in GDSX.

If server logic is kept in servers that execute under the Pathway system and are placed in different CPUs from the GDSX process, the GDSX process can carry out requester functions for multiple device-handler threads without delays for the execution of server logic on behalf of an individual thread. In addition, under the Pathway system, the PATHMON and LINKMON processes provide dynamic link-management for servers accessed through TCPs and through the Pathsend facility. At times of high activity, server processes can be added as needed in different CPUs; requester or device-handler messages can be sent to servers in different CPUs for load balancing, and there is an autorestart mechanism for servers.

In some special cases, there can be advantages to placing server functions in a GDSX process. For example, the Pathway requester-server configuration in Figure 9 is less efficient than the GDSX-Pathsend configuration in Figure 10. In Figure 9, a Pathway requester first sends data to the check server. The check server evaluates (checks) the data, determines which transaction server the requester should send the data to, and returns this information to the requester. Finally, the requester sends the data to the specified server. This sequence costs two interprocess messages and the additional processing time needed for the SCREEN COBOL requester and the two Pathway servers.

Figure 10 shows a GDSX process that contains both requester and check-server functions. Within the GDSX process, a set of device-handler threads can function as requesters for I/O devices. A separate special-purpose device-handler task can function as a check server for the device-handler requester threads. In this case, the GDSX process provides its own intertask communication between the requester threads and the special-purpose task, and only one message is sent between the GDSX process and the transaction server.

Managing I/O Devices Through the GDSX-SCF Interface

GDSX processes maintain their own tables for monitoring the status of GDSX subdevices (device-handler threads) and GDSX lines (line-handler threads). The GDSX interface to SCF commands allows an operator or application to use SCF informational commands such as INFO and STATUS to monitor the states of I/O devices associated with GDSX threads and to issue SCF commands such as STOP, START, ABORT, and ALTER, that can change the state of an I/O device. Figure 11 shows the use of the GDSX-SCF interface to manage workstations.

In Figure 11, an operator or a network management application issues SCF commands. The SCF process directs the commands to the Subsystem Control Point (SCP) process, which forwards commands to data communications subsystems. The SCP process sends the SCF commands to the GDSX process, which applies the commands to the appropriate threads. Since workstations are represented as threads within the GDSX process, checking the status of all workstations and issuing commands against them only requires sending messages to one GDSX process. Without a GDSX process, if each workstation counted as a separate process, checking the status of all workstations and issuing commands against them would require the time-consuming operation of sending separate messages to every process representing a workstation.

Conclusion

GDSX-supplied code provides multitasking and other features useful for developing front-end and back-end processes. In a GDSX front-end process, a user-coded device handler is the basis for device-handler threads that manage the input from I/O devices and provide functions such as data-stream conversion, implementation of a communication protocol, and network-communication error handling. In a back-end process, multiple Tandem host processes communicate with a limited number of I/O devices. Common uses of a GDSX back-end process include implementation of nonstandard communication protocols, message switching, and coordination of access to shared resources or I/O devices, such as log files, terminals, and remote ports.

D-series releases of GDSX contain important features not available with C-series GDSX software: access to Pathsend servers, TMF protection within GDSX processes, and enhanced intertask messaging for threads in a GDSX process.

Figure 11

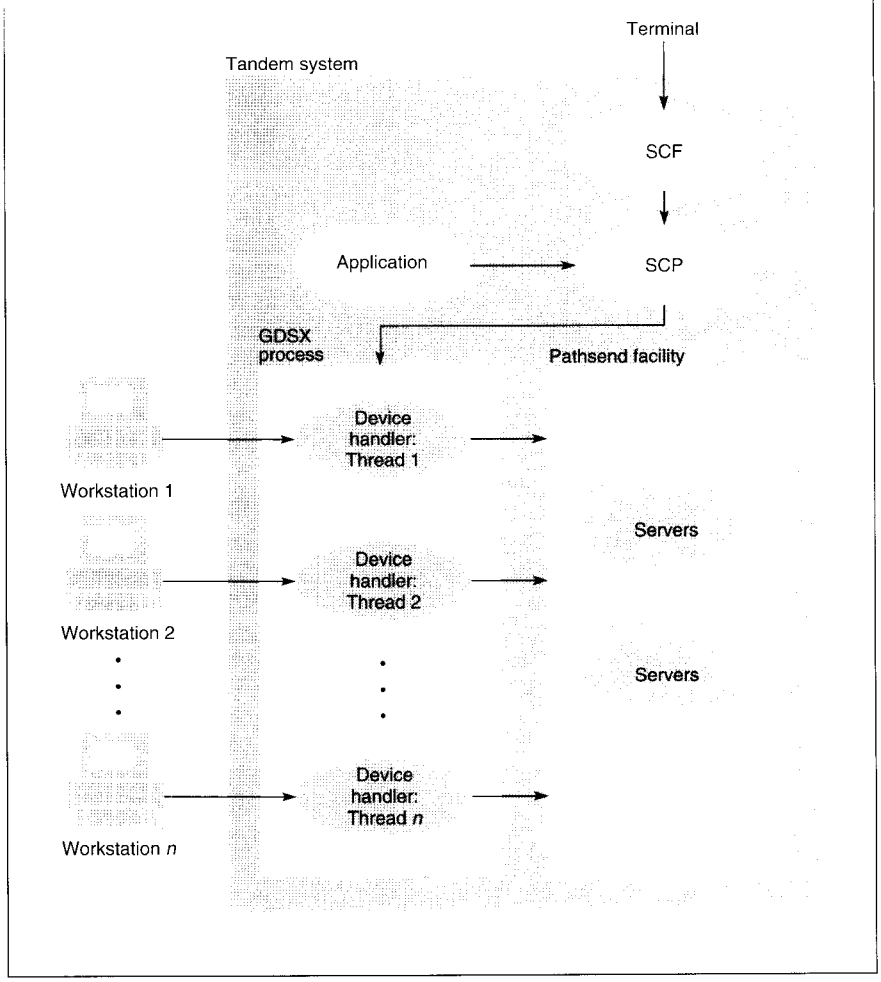
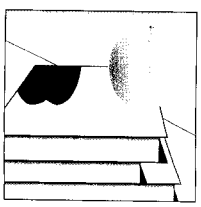


Figure 11. *Managing workstations through the GDSX-SCF interface.*

References

Extended General Device Support (GDSX) Manual. 1993. Tandem Computers Incorporated. Part no. 95805.

Andreas Hotea joined Tandem Austria as a systems analyst in 1988. In 1991 he became a project manager with Tandem Germany and then, later in the year, transferred to the United States as a developer for the GDSX product. In 1993, Andreas became the software development manager for several Pathway products, including GDSX.



Tandem Education

The following paragraphs provide highlights of the latest education courses offered by Tandem. To sign up for a class or to order an independent study program (ISP), users should call 1-800-621-9198. Full descriptions of all available courses and ISPs appear in the *Tandem Education Course Catalog* and on InfoWay.

UNIX Basics

In this four-day course, students use hands-on practice to acquire basic UNIX skills. These include getting started with UNIX, defining and using the UNIX hierarchy, customizing student startup scripts, and creating simple shell scripts using the vi editor. Students also learn the use of over 60 UNIX and shell utilities, with a choice of the Korm Shell or C-shell, on an Integrity SVR4 platform.

Integrity CM-1300 Hardware Installation

This videotape demonstrates the installation and checkout of the Integrity CM-1300 system. The tape runs for approximately 30 minutes.

3217/5289 Data Path Adapter Subsystem

This independent study program includes a 30-minute videotape and a student quiz. The Data Path Adapter interconnects the Tandem Guardian system with the STK automatic cartridge tape handling system. This program introduces the Tandem 3217/5289 Data Path Adapter subsystem and demonstrates the hardware installation and cabling procedures. The Tandem Maintenance and Diagnostic System (TMDS) and troubleshooting techniques are also discussed.

NonStop CO-CLX800 and CO-Cyclone/R Installation

This training aid kit consists of a high-level product guide and a videotape that shows a typical installation of a NEBS cabinet. The training aid gives a general description of the NEBS-compliance CO-CLX800 and CO-Cyclone/R systems, and provides the information needed to service the NEBS hardware by introducing the student to the new or changed components of this system.

The Technical Information and Education department is an annotated list of new Tandem education courses and consulting and information services, as well as other technical information of interest to Tandem users.

Remote Server Call (RSC) Operations

This one-day course gives students an overview of Tandem's Remote Server Call (RSC) product. RSC allows workstations to access Pathway servers and other operating system processes on the Tandem host system. Students learn the installation and configuration procedures required to set up an RSC environment. Students also participate in a discussion of the best practices for managing an RSC environment.

Remote Server Call (RSC) Programming

Tandem's Remote Server Call (RSC) product allows workstations to access Pathway servers and other operating system processes on the Tandem host system. Much of RSC programming is client-related. This three-day classroom-and-lab course provides the knowledge and skills needed to develop RSC-based applications. The course labs give students an immediate understanding of programming in the RSC environment. Students code an RSC client application, using most of the Application Programming Interface (API) calls. Students also experiment with Unsolicited Message Service (UMS) calls and with an Access Control Server (ACS).

Tandem Operations for the Automated Cartridge System

This video program teaches how to configure and operate the Tandem interface to an attached STK 4400 Automated Cartridge System. This program lasts less than an hour and is ideal as a survey or refresher course.

Install Program Simulator

This course uses computer-based training (CBT) running in the Microsoft Windows environment. It allows students to become familiar with the Install program dialog and provides helpful tips for each phase. It features hands-on experience through the use of a workstation-based Install simulator. The course provides a tutorial on the user dialog and gives tips on how to save time and disk space during the installation process. This course also provides the opportunity to run each phase with automatic error correction and with a challenging exercise.

Install Program Overview

In this course, using computer-based training (CBT) running in the Microsoft Windows environment, students become familiar with the functionality and operation of the Install program on Tandem Guardian systems. The course provides an overview of the software installation illustrated with computer animation. The course also describes important components of the Install program and includes a review designed to help students apply what they have learned.

Migrating to the NonStop Kernel

This four-day course is designed especially for those who are thinking of migrating to the D20 release of the Guardian system. Students learn how current and new Tandem products can benefit from D-series enhancements. The course enables students to determine whether they need to modify their systems prior to upgrading, how to convert their applications, and how to use D-series extensions.

OSI/FTAM Concepts and Facilities

This classroom-and-lab course presents detailed technical information about OSI/FTAM, including configuration, system management, programming, and troubleshooting OSI/FTAM subsystems. The course lasts four days.

Syshealth Toolkit

This course uses a new training method, Audiodigital Technology (ADT), to teach students how to use the Syshealth Toolkit, which monitors system resources in the Tandem environment. The course walks students through the processes of installation, configuration, remote notification, and operations. By the end of the course, students are familiar with the capabilities of the Syshealth Toolkit and how it can help them administer their Tandem installations.

Dynamic System Configuration (DSC)

Intended for system managers and operators, this Audiodigital Technology (ADT) course provides an informative introduction to Dynamic System Configuration and its user interface Configuration Utility Program (COUP). Upon completion of the course, students understand how to add, start, stop, or alter devices, controllers, and paths online without doing a SYSGEN.

Tandem Education Training

Audiodigital Technology

Tandem Education has introduced a powerful new training method, Audiodigital Technology (ADT). With ADT, the student sits at a workstation and watches as a Tandem application is demonstrated by an expert on screen. The accompanying audio guides the student through the course. At any point, the student can fast-forward or rewind the course, and the video and audio portions remain perfectly synchronized. In addition, the student can exit the training mode and instantly access the actual application to practice the skills already learned.

Among the benefits of ADT are the following:

- It is very easy to use. Anyone who can operate a cassette player can use ADT.
- It has a low cost of ownership. The ADT player includes all software and a run-time license for unlimited courseware.
- It is cost effective. One ADT course and player can be used by an unlimited number of students. In addition, Tandem Education is constantly releasing new ADT courses.

To run an ADT course, the workstation used must be a 286 or higher IBM PC, or compatible, with an available com1 or com2 port. It must also have a color display (CGA, EGA, or VGA) and an internal hard disk. All Tandem ADT courses also require a QTrain player. For price and ordering information on the QTrain player, users should contact User Training Services Group at 1-800-395-9991 (phone) or 415-322-0528 (fax).

TandemSystemsReview*Index*

The *Tandem Journal* became the *Tandem Systems Review* in February 1985. Four issues of the *Tandem Journal* were published:

Volume 1, No. 1	Fall 1983
Volume 2, No. 1	Winter 1984
Volume 2, No. 2	Spring 1984
Volume 2, No. 3	Summer 1984

As of this issue, 24 issues of the *Tandem Systems Review* have been published:

Volume 1, No. 1	Feb. 1985	Volume 6, No. 1	March 1990
Volume 1, No. 2	June 1985	Volume 6, No. 2	Oct. 1990
Volume 2, No. 1	Feb. 1986	Volume 7, No. 1	April 1991
Volume 2, No. 2	June 1986	Volume 7, No. 2	Oct. 1991
Volume 2, No. 3	Dec. 1986	Volume 8, No. 1	Spring 1992
Volume 3, No. 1	March 1987	Volume 8, No. 2	Summer 1992
Volume 3, No. 2	Aug. 1987	Volume 8, No. 3	Fall 1992
Volume 4, No. 1	Feb. 1988	Volume 9, No. 1	Winter 1993
Volume 4, No. 2	July 1988	Volume 9, No. 2	Spring 1993
Volume 4, No. 3	Oct. 1988	Volume 9, No. 3	Summer 1993
Volume 5, No. 1	April 1989	Volume 9, No. 4	Fall 1993
Volume 5, No. 2	Sept. 1989	Volume 10, No. 1	Jan. 1994

The articles published in all 28 issues are arranged by subject below. (*Tandem Journal* is abbreviated as TJ and *Tandem Systems Review* as TSR.) A second index, arranged by product, is also provided.

Index by Subject

Article title	Author(s)	Publication	Volume, Issue	Publication date	Part number
APPLICATION DEVELOPMENT AND LANGUAGES					
A New Design for the PATHWAY TCP	R. Wong	TJ	2,2	Spring 1984	83932
An Overview of Client/Server Computing on Tandem Systems	H. Cooperstein	TSR	8,3	Fall 1992	89803
An Introduction to Tandem EXTENDED BASIC	J. Meyerson	TJ	2,2	Spring 1984	83932
Application Code Conversion for D-Series Systems	K. Liu	TSR	9,2	Spring 1993	89805
Application Profile: Storing Macintosh Graphics on the Tandem 5200 Optical Storage Facility	D. Broyles	TSR	9,3	Summer 1993	89806
Basic Uses and New Features of Extended GDS	A. Hotea	TSR	10,1	Jan. 1994	104396
Debugging TACL Code	L. Palmer	TSR	4,2	July 1988	13693
Designing and Implementing a Graphical User Interface	S. Wolfe	TSR	9,3	Summer 1993	89806
Designing Client/Server Applications for OLTP on Guardian 90 Systems	W. Culman	TSR	8,3	Fall 1992	89803
Extending the Client/Server Model With Object-Oriented Technology	T. Rohner	TSR	10,1	Jan. 1994	104396
Implementing Client/Server Using RSC	M. Iem, T. Kocher	TSR	8,3	Fall 1992	89803
Instrumenting Applications for Effective Event Management	J. Dagenais	TSR	7,2	Oct. 1991	65248
New TAL Features	C. Lu, J. Murayama	TSR	2,2	June 1986	83837
PATHFINDER–An Aid for Application Development	S. Benett	TJ	1,1	Fall 1983	83930

Article title	Author(s)	Publication	Volume, Issue	Publication date	Part number
APPLICATION DEVELOPMENT AND LANGUAGES *(cont.)*					
PATHWAY IDS: A Message-level Interface to Devices and Processes	M. Anderton, M. Noonan	TSR	2,2	June 1986	83937
The RESPOND OLTP Business Management System for Manufacturing	H. Bolling, W. Bronson	TSR	9,1	Winter 1993	89804
State-of-the-Art C Compiler	E. Kit	TSR	2,2	June 1986	83937
TACL, Tandem's New Extensible Command Language	J. Campbell, R. Glascock	TSR	2,1	Feb. 1986	83936
Tandem's New COBOL85	D. Nelson	TSR	2,1	Feb. 1986	83936
The DAL Server: Client/Server Access to Tandem Databases	W. Schlansky, J. Schrengohst	TSR	9,1	Winter 1993	89804
The ENABLE Program Generator for Multifile Applications	B. Chapman, J. Zimmerman	TSR	1,1	Feb. 1985	83934
TMF and the Multi-Threaded Requester	T. Lemberger	TJ	1,1	Fall 1983	83930
Writing a Command Interpreter	D. Wong	TSR	1,2	June 1985	83935
CLIENT/SERVER					
An Overview of Client/Server Computing on Tandem Systems	H. Cooperstein	TSR	8,3	Fall 1992	89803
Application Profile: Storing Macintosh Graphics on the Tandem 5200 Optical Storage Facility	D. Broyles	TSR	9,3	Summer 1993	89806
Designing and Implementing a Graphical User Interface	S. Wolfe	TSR	9,3	Summer 1993	89806
Designing Client/Server Applications for OLTP on Guardian 90 Systems	W. Culman	TSR	8,3	Fall 1992	89803
Extending the Client/Server Model With Object-Oriented Technology	T. Rohner	TSR	10,1	Jan. 1994	104396
Gateways to NonStop SQL	D. Slutz	TSR	6,2	Oct. 1990	46987
Implementing Client/Server Using RSC	M. Iem, T. Kocher	TSR	8,3	Fall 1992	89803
The DAL Server: Client/Server Access to Tandem Databases	W. Schlansky, J. Schrengohst	TSR	9,1	Winter 1993	89804
DATA COMMUNICATIONS					
An Overview of SNAX/CDF	M. Turner	TSR	5,2	Sept. 1989	28152
A SNAX Passthrough Tutorial	D. Kirk	TJ	2,2	Spring 1984	83932
Basic Uses and New Features of Extended GDS	A. Hotea	TSR	10,1	Jan. 1994	104396
Changes in FOX	N. Donde	TSR	1,2	June 1985	83935
Connecting Terminals and Workstations to Guardian 90 Systems	E. Siegel	TSR	8,2	Summer 1992	69848
Expand High-Performance Solutions	D. Smith	TSR	9,3	Summer 1993	89806
Introduction to MULTILAN	A. Coyle	TSR	4,1	Feb. 1988	11078
Overview of the MULTILAN Server	A. Rowe	TSR	4,1	Feb. 1988	11078
SNAX/APC: Tandem's New SNA Software for Distributed Processing	B. Grantham	TSR	3,1	March 1987	83939
SNAX/HLS: An Overview	S. Saltwick	TSR	1,2	June 1985	83935
TLAM: A Connectivity Option for Expand	K. MacKenzie	TSR	7,1	April 1991	46988
Using the MULTILAN Application Interfaces	M. Berg, A. Rowe	TSR	4,1	Feb. 1988	11078

Article title	Author(s)	Publication	Volume, Issue	Publication date	Part number
DATA MANAGEMENT					
A Comparison of the B00 DP1 and DP2 Disc Processes	T. Schachter	TSR	1,2	June 1985	83935
An Overview of NonStop SQL Release 2	M. Pong	TSR	6,2	Oct. 1990	46987
Batch Processing in Online Enterprise Computing	T. Keefauver	TSR	6,2	Oct. 1990	46987
Concurrency Control Aspects of Transaction Design	W. Senf	TSR	6,1	March 1990	32968
Converting Database Files from ENSCRIBE to NonStop SQL	W. Weikel	TSR	6,1	March 1990	32986
DP1-DP2 File Conversion: An Overview	J. Tate	TSR	2,1	Feb. 1986	83936
Determining FCP Conversion Time	J. Tate	TSR	2,1	Feb. 1986	83936
DP2's Efficient Use of Cache	T. Schachter	TSR	1,2	June 1985	83935
DP2 Highlights	K. Carlyle, L. McGowan	TSR	1,2	June 1985	83935
DP2 Key-sequenced Files	T. Schachter	TSR	1,2	June 1985	83935
Gateways to NonStop SQL	D. Slutz	TSR	6,2	Oct. 1990	46987
High-Performance SQL Through Low-Level System Integration	A. Borr	TSR	4,2	July 1988	13693
Improvements in TMF	T. Lemberger	TSR	1,2	June 1985	83935
NetBatch: Managing Batch Processing on Tandem Systems	D. Wakashige	TSR	5,1	April 1989	18662
NetBatch-Plus: Structuring the Batch Environment	G. Earle, D. Wakashige	TSR	6,1	March 1990	32986
NonStop SQL: The Single Database Solution	J. Cassidy, T. Kocher	TSR	5,2	Sept. 1989	28152
NonStop SQL Data Dictionary	R. Holbrook, D. Tsou	TSR	4,2	July 1988	13693
NonStop SQL Optimizer: Basic Concepts	M. Pong	TSR	4,2	July 1988	13693
NonStop SQL Optimizer: Query Optimization and User Influence	M. Pong	TSR	4,2	July 1988	13693
NonStop SQL Reliability	C. Fenner	TSR	4,2	July 1988	13693
Online Information Processing	J. Viescas	TSR	9,1	Winter 1993	89804
Online Reorganization of Key-Sequenced Tables and Files	G. Smith	TSR	6,2	Oct. 1990	46987
Optimizing Batch Performance	T. Keefauver	TSR	5,2	Sept. 1989	28152
Overview of NonStop SQL	H. Cohen	TSR	4,2	July 1988	13693
Parallelism in NonStop SQL Release 2	M. Moore, A. Sodhi	TSR	6,2	Oct. 1990	46987
The NonStop SQL Release 2 Benchmark	S. Englert, J. Gray, T. Kocher, P. Shah	TSR	6,2	Oct. 1990	46987
The Outer Join in NonStop SQL	J. Vaishnav	TSR	6,2	Oct. 1990	46987
The Relational Data Base Management Solution	G. Ow	TJ	2,1	Winter 1984	83931
Tandem's NonStop SQL Benchmark	Tandem Performance Group	TSR	4,1	Feb. 1988	11078
The TRANSFER Delivery System for Distributed Applications	S. Van Pelt	TJ	2,2	Spring 1984	83932
TMF Autorollback: A New Recovery Feature	M. Pong	TSR	1,1	Feb. 1985	83934
OBJECT-ORIENTED TECHNOLOGY					
Extending the Client/Server Model With Object-Oriented Technology	T. Rohner	TSR	10,1	Jan. 1994	104396

Article title	Author(s)	Publication	Volume, Issue	Publication date	Part number
OPERATING SYSTEMS					
Application Code Conversion for D-Series Systems	K. Liu	TSR	9,2	Spring 1993	89805
Highlights of the B00 Software Release	K. Coughlin, R. Montevaldo	TSR	1,2	June 1985	83935
Increased Code Space	A. Jordan	TSR	1,2	June 1985	83935
Managing System Time Under GUARDIAN 90	E. Nellen	TSR	2,1	Feb. 1986	83936
Migration Planning for D-Series Systems	S. Kuukka	TSR	9,2	Spring 1993	89805
New GUARDIAN 90 Time-keeping Facilities	E. Nellen	TSR	1,2	June 1985	83935
New Process-timing Features	S. Sharma	TSR	1,2	June 1985	83935
NonStop II Memory Organization and Extended Addressing	D. Thomas	TJ	1,1	Fall 1983	83930
Overview of the C00 Release	L. Marks	TSR	4,1	Feb. 1988	11078
Overview of the D-Series Guardian 90 Operating System	W. Bartlett	TSR	9,2	Spring 1993	89805
Overview of the NonStop-UX Operating System for the Integrity S2	P. Norwood	TSR	7,1	April 1991	46988
Robustness to Crash in a Distributed Data Base: A Nonshared-memory Approach	A. Borr	TSR	1,2	June 1985	83935
The GUARDIAN Message System and How to Design for It	M. Chandra	TSR	1,1	Feb. 1985	83935
The Tandem Global Update Protocol	R. Carr	TSR	1,2	June 1985	83935
PERFORMANCE AND CAPACITY PLANNING					
A Performance Retrospective	P. Oleinick, P. Shah	TSR	2,3	Dec. 1986	83938
Buffering for Better Application Performance	R. Mattran	TSR	2,1	Feb. 1986	83936
Capacity Planning Concepts	R. Evans	TSR	2,3	Dec. 1986	83938
Capacity Planning With TCM	W. Highleyman	TSR	7,2	Oct. 1991	65248
C00 TMDS Performance	J. Mead	TSR	4,1	Feb. 1988	11078
Credit-authorization Benchmark for High Performance and Linear Growth	T. Chmiel, T. Houy	TSR	2,1	Feb. 1986	83936
Debugging Accelerated Programs on TNS/R Systems	D. Cressler	TSR	8,1	Spring 1992	65250
DP2 Performance	J. Enright	TSR	1,2	June 1985	83935
Estimating Host Response Time in a Tandem System	H. Horwitz	TSR	4,3	Oct. 1988	15748
Expand High-Performance Solutions	D. Smith	TSR	9,3	Summer 1993	89806
FASTSORT: An External Sort Using Parallel Processing	J. Gray, M. Stewart, A. Tsukerman, S. Uren, B. Vaughan	TSR	2,3	Dec. 1986	83938
Getting Optimum Performance from Tandem Tape Systems	A. Khatri	TSR	2,3	Dec. 1986	83938
How to Set Up a Performance Data Base with MEASURE and ENFORM	M. King	TSR	2,3	Dec. 1986	83938
Implementing a Systems Management Improvement Program	J. Dagenais	TSR	9,4	Fall 1993	89807
Improved Performance for BACKUP2 and RESTORE2	A. Khatri, M. McCline	TSR	1,2	June 1985	83935
Improving Performance on TNS/R Systems With the Accelerator	M. Blanchet	TSR	8,1	Spring 1992	65250
MEASURE: Tandem's New Performance Measurement Tool	D. Dennison	TSR	2,3	Dec. 1986	83938
Measuring DSM Event Management Performance	M. Stockton	TSR	8,1	Spring 1992	65250
Message System Performance Enhancements	D. Kinkade	TSR	2,3	Dec. 1986	83938
Message System Performance Tests	S. Uren	TSR	2,3	Dec. 1986	83938
Network Design Considerations	J. Evjen	TSR	5,2	Sept. 1989	28152
NonStop NET/MASTER: Configuration and Performance Guidelines	M. Stockton	TSR	9,4	Fall 1993	89807
NonStop VLX Performance	J. Enright	TSR	2,3	Dec. 1986	83938
Optimizing Sequential Processing on the Tandem System	R. Welsh	TJ	2,3	Summer 1984	83933
Pathway TCP Enhancements for Application Run-Time Support	R. Vannucci	TSR	7,1	April 1991	46988

Article title	Author(s)	Publication	Volume, Issue	Publication date	Part number
PERFORMANCE AND CAPACITY PLANNING *(cont.)*					
Performance Benefits of Parallel Query Execution and Mixed Workload Support in NonStop SQL Release 2	S. Englert, J. Gray	TSR	6,2	Oct. 1990	46987
Performance Considerations for Application Processes	R. Glasstone	TSR	2,3	Dec. 1986	83938
Performance Measurements of an ATM Network Application	N. Cabell, D. Mackie	TSR	2,3	Dec. 1986	83938
Predicting Response Time in On-line Transaction Processing Systems	A. Khatri	TSR	2,2	June 1986	83937
The 6600 and TCC6820 Communications Controllers: A Performance Comparison	P. Beadles	TSR	2,3	Dec. 1986	83938
The ENCORE Stress Test Generator for On-line Transaction Processing Applications	S. Kosinski	TJ	2,1	Winter 1984	83931
The PATHWAY TCP: Performance and Tuning	J. Vatz	TSR	1,1	Feb. 1985	83934
The Performance Characteristics of Tandem NonStop Systems	J. Day	TJ	1,1	Fall 1983	83930
Sizing Cache for Applications that Use B-series DP1 and TMF	P. Shah	TSR	2,2	June 1986	83937
Sizing the Spooler Collector Data File	H. Norman	TSR	4,1	Feb. 1988	11978
Tandem's 5200 Optical Storage Facility: Performance and Optimization Considerations	S. Coleman	TSR	5,1	April 1989	18662
Tandem's Approach to Fault Tolerance	B. Ball, W. Bartlett, S. Thompson	TSR	4,1	Feb. 1988	11078
Understanding PATHWAY Statistics	R. Wong	TJ	2,2	Spring 1984	83932
PERIPHERALS					
5120 Tape Subsystem Recording Technology	W. Phillips	TSR	3,2	Aug. 1987	83940
An Introduction to DYNAMITE Workstation Host Integration	S. Kosinski	TSR	1,2	June 1985	83935
Application Profile: Storing Macintosh Graphics on the Tandem 5200 Optical Storage Facility	D. Broyles	TSR	9,3	Summer 1993	89806
Data-Encoding Technology Used in the XL8 Storage Facility	D. S. Ng	TSR	2,2	June 1986	83937
Data-Window Phase-Margin Analysis	A. Painter, H. Pham, H. Thomas	TSR	2,2	June 1986	83937
Introducing the 3207 Tape Controller	S. Chandran	TSR	1,2	June 1985	83935
Peripheral Device Interfaces	J. Blakkan	TSR	3,2	Aug. 1987	83940
Plated Media Technology Used in the XL8 Storage Facility	D.S. Ng	TSR	2,2	June 1986	83937
Streaming Tape Drives	J. Blakkan	TSR	3,2	Aug. 1987	83940
Terminal Selection	E. Siegel	TSR	8,2	Summer 1992	69848
The 5200 Optical Storage Facility: A Hardware Perspective	A. Patel	TSR	5,1	April 1989	18662
The 6100 Communications Subsystem: A New Architecture	R. Smith	TJ	2,1	Winter 1984	83931
The 6600 and TCC6820 Communications Controllers: A Performance Comparison	P. Beadles	TSR	2,3	Dec. 1986	83938
The DYNAMITE Workstation: An Overview	G. Smith	TSR	1,2	June 1985	83935
The Model 6VI Voice Input Option: Its Design and Implementation	B. Huggett	TJ	2,3	Summer 1984	83933
The Role of Optical Storage in Information Processing	L. Sabaroff	TSR	3,2	Aug. 1987	83940
The V8 Disc Storage Facility: Setting a New Standard for On-line Disc Storage	M. Whiteman	TSR	1,2	June 1985	83935
PROCESSORS					
Fault Tolerance in the NonStop Cyclone System	S. Chan, R. Jardine	TSR	7,1	April 1991	46988
A Hardware Overview of the NonStop Himalaya K10000 Server	C. Kong	TSR	10,1	Jan. 1994	104396
NonStop CLX: Optimized for Distributed On-Line Transaction Processing	D. Lenoski	TSR	5,1	April 1989	18662
NonStop VLX Hardware Design	M. Brown	TSR	2,3	Dec. 1986	83938
Overview of Tandem NonStop Series/RISC Systems	L. Faby, R. Mateosian	TSR	8,1	Spring 1992	65250
The High-Performance NonStop TXP Processor Transaction Processing	W. Bartlett, T. Houy, D. Meyer	TJ	2,1	Winter 1984	83931
The NonStop TXP Processor: A Powerful Design for On-line Transaction Processing	P. Oleinick	TJ	2,3	Summer 1984	83933
The VLX: A Design for Serviceability	J. Allen, R. Boyle	TSR	3,1	March 1987	83939

Article title	Author(s)	Publication	Volume, Issue	Publication date	Part number
SECURITY					
Dial-In Security Considerations	P. Grainger	TSR	7,2	Oct. 1991	65248
Distributed Protection with SAFEGUARD	T. Chou	TSR	2,2	June 1986	83937
Enhancing System Security With Safeguard	C. Gaydos	TSR	7,1	April 1991	46988
SYSTEM CONNECTIVITY					
Basic Uses and New Features of Extended GDS	A. Hotea	TSR	10,1	Jan. 1994	104396
Building Open Systems Interconnection with OSI/AS and OSI/TS	R. Smith	TSR	6,1	March 1990	32986
Connecting Terminals and Workstations to Guardian 90 Systems	E. Siegel	TSR	8,2	Summer 1992	69848
Implementing Client/Server Using RSC	M. Iem, T. Kocher	TSR	8,3	Fall 1992	89803
Network Design Considerations	J. Evjen	TSR	5,2	Sept. 1989	28152
Terminal Connection Alternatives for Tandem Systems	J. Simonds	TSR	5,1	April 1989	18662
Terminal Selection	E. Siegel	TSR	8,2	Summer 1992	69848
The OSI Model: Overview, Status, and Current Issues	A. Dunn	TSR	5,1	April 1989	18662
SYSTEM MANAGEMENT					
Configuring Tandem Disk Subsystems	S. Sitler	TSR	2,3	Dec. 1986	83938
Data Replication in Tandem's Distributed Name Service	T. Eastep	TSR	4,3	Oct. 1988	15748
Enhancements to TMDS	L. White	TSR	3,2	Aug. 1987	83940
Event Management Service Design and Implementation	H. Jordan, R. McKee, R. Schuet	TSR	4,3	Oct. 1988	15748
Implementing a Systems Management Improvement Program	J. Dagenais	TSR	9,4	Fall 1993	89807
Instrumenting Applications for Effective Event Management	J. Dagenais	TSR	7,2	Oct. 1991	65248
Introducing TMDS, Tandem's New On-line Diagnostic System	J. Troisi	TSR	1,2	June 1985	83935
Measuring DSM Event Management Performance	M. Stockton	TSR	8,1	Spring 1992	65250
Network Statistics System	M. Miller	TSR	4,3	Oct. 1988	15748
NonStop NET/MASTER: Configuration and Performance Guidelines	M. Stockton	TSR	9,4	Fall 1993	89807
NonStop NET/MASTER: Event Management Architecture	M. Stockton	TSR	9,4	Fall 1993	89807
NonStop NET/MASTER: Event Processing Costs and Sizing Calculations	M. Stockton	TSR	9,4	Fall 1993	89807
Overview of DSM	P. Homan, B. Malizia, E. Reisner	TSR	4,3	Oct. 1988	15748
SCP and SCF: A General Purpose Implementation of the Subsystem Programmatic Interface	T. Lawson	TSR	4,3	Oct. 1988	15748
RDF: An Overview	J. Guerrero	TSR	7,2	Oct. 1991	65248
RDF Synchronization	F. Jongma, W. Senf	TSR	8,2	Summer 1992	69848
Tandem's Subsystem Programmatic Interface	G. Tom	TSR	4,3	Oct. 1988	15748
Using FOX to Move a Fault-tolerant Application	C. Breighner	TSR	1,1	Feb. 1985	83934
Using the Subsystem Programmatic Interface and Event Management Services	K. Stobie	TSR	4,3	Oct. 1988	15748
VIEWPOINT Operations Console Facility	R. Hansen, G. Stewart	TSR	4,3	Oct. 1988	15748
VIEWSYS: An On-line System-resource Monitor	D. Montgomery	TSR	1,2	June 1985	83935
Writing Rules for Automated Operations	J. Collins	TSR	7,2	Oct. 1991	65248
UTILITIES					
Enhancements to PS MAIL	R. Funk	TSR	3,1	March 1987	83939

Index by Product

Article title	Author(s)	Publication	Volume, Issue	Publication date	Part number
3207 TAPE CONTROLLER					
Introducing the 3207 Tape Controller	S. Chandran	TSR	1,2	June 1985	83935
5120 TAPE SUBSYSTEM					
5120 Tape Subsystem Recording Technology	W. Phillips	TSR	3,2	Aug. 1987	83940
5200 OPTICAL STORAGE					
Application Profile: Storing Macintosh Graphics on the Tandem 5200 Optical Storage Facility	D. Broyles	TSR	9,3	Summer 1993	89806
Tandem's 5200 Optical Storage Facility: Performance and Optimization Considerations	S. Coleman	TSR	5,1	April 1989	18662
The 5200 Optical Storage Facility: A Hardware Perspective	A. Patel	TSR	5,1	April 1989	18662
The Role of Optical Storage in Information Processing	L. Sabaroff	TSR	4,1	Feb. 1988	11078
6100 COMMUNICATIONS SUBSYSTEM					
The 6100 Communications Subsystem: A New Architecture	R. Smith	TJ	2,1	Winter 1984	83931
6530 TERMINAL					
The Model 6VI Voice Input Option: Its Design and Implementation	B. Huggett	TJ	2,3	Summer 1984	83933
6600 AND TCC6820 COMMUNICATIONS CONTROLLERS					
The 6600 and TCC6820 Communications Controllers: A Performance Comparison	P. Beadles	TSR	2,3	Dec. 1986	83938
BASIC					
An Introduction to Tandem EXTENDED BASIC	J. Meyerson	TJ	2,2	Spring 1984	83932
C					
State-of-the-art C Compiler	E. Kit	TSR	2,2	June 1986	83937
CIS					
Customer Information Service	J. Massucco	TSR	3,1	March 1987	83939
CLX					
NonStop CLX: Optimized for Distributed On-Line Transaction Processing	D. Lenoski	TSR	5,1	April 1989	18662
COBOL85					
Tandem's New COBOL85	D. Nelson	TSR	2,1	Feb. 1986	83936
COMINT (CI)					
Writing a Command Interpreter	D. Wong	TSR	1,2	June 1985	83935
CYCLONE					
Fault Tolerance in the NonStop Cyclone System	S. Chan, R. Jardine	TSR	7,1	April 1991	46988
DAL SERVER					
The DAL Server: Client/Server Access to Tandem Databases	W. Schlansky, J. Schrengohst	TSR	9,1	Winter 1993	89804

Article title	Author(s)	Publication	Volume, Issue	Publication date	Part number
DP1 AND DP2					
A Comparison of the B00 DP1 and DP2 Disc Processes	T. Schachter	TSR	1,2	June 1985	83935
Determining FCP Conversion Time	J. Tate	TSR	2,1	Feb. 1986	83936
DP1-DP2 File Conversion: An Overview	J. Tate	TSR	2,1	Feb. 1986	83936
DP2 Highlights	K. Carlyle, L. McGowan	TSR	1,2	June 1985	83935
DP2 Key-sequenced Files	T. Schachter	TSR	1,2	June 1985	83935
DP2 Performance	J. Enright	TSR	1,2	June 1985	83935
DP2's Efficient Use of Cache	T. Schachter	TSR	1,2	June 1985	83935
Sizing Cache for Applications that Use B-series DP1 and TMF	P. Shah	TSR	2,2	June 1986	83937
DSM					
Data Replication in Tandem's Distributed Name Service	T. Eastep	TSR	4,3	Oct. 1988	15748
Event Management Service Design and Implementation	H. Jordan, R. McKee, R. Schuet	TSR	4,3	Oct. 1988	15748
Instrumenting Applications for Effective Event Management	J. Dagenais	TSR	7,2	Oct. 1991	65248
Measuring DSM Event Management Performance	M. Stockton	TSR	8,1	Spring 1992	65250
Network Statistics System	M. Miller	TSR	4,3	Oct. 1988	15748
Overview of DSM	P. Homan, B. Malizia, E. Reisner	TSR	4,3	Oct. 1988	15748
SCP and SCF: A General Purpose Implementation of the Subsystem Programmatic Interface	T. Lawson	TSR	4,3	Oct. 1988	15748
Tandem's Subsystem Programmatic Interface	G. Tom	TSR	4,3	Oct. 1988	15748
Using the Subsystem Programmatic Interface and Event Management Services	K. Stobie	TSR	4,3	Oct. 1988	15748
VIEWPOINT Operations Console Facility	R. Hansen, G. Stewart	TSR	4,3	Oct. 1988	15748
Writing Rules for Automated Operations	J. Collins	TSR	7,2	Oct. 1991	65248
DYNAMITE					
An Introduction to DYNAMITE Workstation Host Integration	S. Kosinski	TSR	1,2	June 1985	83935
The DYNAMITE Workstation: An Overview	G. Smith	TSR	1,2	June 1985	83935
ENABLE					
The ENABLE Program Generator for Multifile Applications	B. Chapman, J. Zimmerman	TSR	1,1	Feb. 1985	83934
ENCOMPASS					
The Relational Data Base Management Solution	G. Ow	TJ	2,1	Winter 1984	83931
ENCORE					
The ENCORE Stress Test Generator for On-line Transaction Processing Applications	S. Kosinski	TJ	2,1	Winter 1984	83931
ENSCRIBE					
Converting Database Files from ENSCRIBE to NonStop SQL	W. Weikel	TSR	6,1	March 1990	32986
EXPAND					
Expand High-Performance Solutions	D. Smith	TSR	9,3	Summer 1993	89806
FASTSORT					
FASTSORT: An External Sort Using Parallel Processing	J. Gray, M. Stewart, A. Tsukerman, S. Uren, B. Vaughan	TSR	2,3	Dec. 1986	83938

Article title	Author(s)	Publication	Volume, Issue	Publication date	Part number
FOX					
Changes in FOX	N. Donde	TSR	1,2	June 1985	83935
Using FOX to Move a Fault-tolerant Application	C. Breighner	TSR	1,1	Feb. 1985	83934
FUP					
Online Reorganization of Key-Sequenced Tables and Files	G. Smith	TSR	6,2	Oct. 1990	46987
GDS					
Basic Uses and New Features of Extended GDS	A. Hotea	TSR	10,1	Jan. 1994	104396
GUARDIAN 90					
Application Code Conversion for D-Series Systems	K. Liu	TSR	9,2	Spring 1993	89805
B00 Software Manuals	S. Olds	TSR	1,2	June 1985	83935
C00 Software Manuals	E. Levi	TSR	4,1	Feb. 1988	11078
Highlights of the B00 Software Release	K. Coughlin, R. Montevaldo	TSR	1,2	June 1985	83935
Improved Performance for BACKUP2 and RESTORE2	A. Khatri, M. McCline	TSR	1,2	June 1985	83935
Increased Code Space	A. Jordan	TSR	1,2	June 1985	83935
Managing System Time Under GUARDIAN 90	E. Nellen	TSR	2,1	Feb. 1986	83936
Message System Performance Enhancements	D. Kinkade	TSR	2,3	Dec. 1986	83938
Message System Performance Tests	S. Uren	TSR	2,3	Dec. 1986	83938
Migration Planning for D-Series Systems	S. Kuukka	TSR	9,2	Spring 1993	89805
New GUARDIAN 90 Time-keeping Facilities	E. Nellen	TSR	1,2	June 1985	83935
New Process-timing Features	S. Sharma	TSR	1,2	June 1985	83935
NonStop II Memory Organization and Extended Addressing	D. Thomas	TJ	1,1	Fall 1983	83930
Overview of the C00 Release	L. Marks	TSR	4,1	Feb. 1988	11078
Overview of the D-Series Guardian 90 Operating System	W. Bartlett	TSR	9,2	Spring 1993	89805
Robustness to Crash in a Distributed Data Base: A Nonshared-memory Multiprocessor Approach	A. Borr	TSR	1,2	June 1985	83935
Tandem's Approach to Fault Tolerance	B. Ball, W. Bartlett, S. Thompson	TSR	4,1	Feb. 1988	11078
The GUARDIAN Message System and How to Design for It	M. Chandra	TSR	1,1	Feb. 1985	83934
The Tandem Global Update Protocol	R. Carr	TSR	1,2	June 1985	83935
HIMALAYA					
A Hardware Overview of the NonStop Himalaya K10000 Server	C. Kong	TSR	10,1	Jan. 1994	104396
INTEGRITY S2					
Overview of the NonStop-UX Operating System for the Integrity S2	P. Norwood	TSR	7,1	April 1991	46988
MEASURE					
How to Set Up a Performance Data Base with MEASURE and ENFORM	M. King	TSR	2,3	Dec. 1986	83938
MEASURE: Tandem's New Performance Measurement Tool	D. Dennison	TSR	2,3	Dec. 1986	83938
MULTILAN					
Introduction to MULTILAN	A. Coyle	TSR	4,1	Feb. 1988	11078
Overview of the MULTILAN Server	A. Rowe	TSR	4,1	Feb. 1988	11078
Using the MULTILAN Application Interfaces	M. Berg, A. Rowe	TSR	4,1	Feb. 1988	11078

Article title	Author(s)	Publication	Volume, Issue	Publication date	Part number
NETBATCH-PLUS					
NetBatch: Managing Batch Processing on Tandem Systems	D. Wakashige	TSR	5,1	April 1989	18662
NetBatch-Plus: Structuring the Batch Environment	G. Earle, D. Wakashige	TSR	6,1	March 1990	32986
NONSTOP NET/MASTER					
NonStop NET/MASTER: Configuration and Performance Guidelines	M. Stockton	TSR	9,4	Fall 1993	89807
NonStop NET/MASTER: Event Management Architecture	M. Stockton	TSR	9,4	Fall 1993	89807
NonStop NET/MASTER: Event Processing Costs and Sizing Calculations	M. Stockton	TSR	9,4	Fall 1993	89807
NONSTOP SQL					
An Overview of NonStop SQL Release 2	M. Pong	TSR	6,2	Oct. 1990	46987
Concurrency Control Aspects of Transaction Design	W. Senf	TSR	6,1	March 1990	32986
Converting Database Files from ENSCRIBE to NonStop SQL	W. Weikel	TSR	6,1	March 1990	32986
Gateways to NonStop SQL	D. Slutz	TSR	6,2	Oct. 1990	46987
High-Performance SQL Through Low-Level System Integration	A. Borr	TSR	4,2	July 1988	13693
NonStop SQL Data Dictionary	R. Holbrook, D. Tsou	TSR	4,2	July 1988	13693
NonStop SQL: The Single Database Solution	J. Cassidy, T. Kocher	TSR	5,2	Sept. 1989	28152
NonStop SQL Optimizer: Basic Concepts	M. Pong	TSR	4,2	July 1988	13693
NonStop SQL Optimizer: Query Optimization and User Influence	M. Pong	TSR	4,2	July 1988	13693
NonStop SQL Reliability	C. Fenner	TSR	4,2	July 1988	13693
Overview of NonStop SQL	H. Cohen	TSR	4,2	July 1988	13693
Parallelism in NonStop SQL Release 2	M. Moore, A. Sodhi	TSR	6,2	Oct. 1990	46987
Performance Benefits of Parallel Query Execution and Mixed Workload Support in NonStop SQL Release 2	S. Englert, J. Gray	TSR	6,2	Oct. 1990	46987
Tandem's NonStop SQL Benchmark	Tandem Performance Group	TSR	4,1	Feb. 1988	11078
The NonStop SQL Release 2 Benchmark	S. Englert, J. Gray, T. Kocher, P. Shah	TSR	6,2	Oct. 1990	46987
The Outer Join in NonStop SQL	J. Vaishnav	TSR	6,2	Oct. 1990	46987
OSI					
Building Open Systems Interconnection with OSI/AS and OSI/TS	R. Smith	TSR	6,1	March 1990	32986
The OSI Model: Overview, Status, and Current Issues	A. Dunn	TSR	5,1	April 1989	18662
PATHFINDER					
PATHFINDER–An Aid for Application Development	S. Benett	TJ	1,1	Fall 1983	83930
PATHWAY					
A New Design for the PATHWAY TCP	R. Wong	TJ	2,2	Spring 1984	83932
PATHWAY IDS: A Message-level Interface to Devices and Processes	M. Anderton, M. Noonan	TSR	2,2	June 1986	83937
Pathway TCP Enhancements for Application Run-Time Support	R. Vannucci	TSR	7,1	April 1991	46988
The PATHWAY TCP: Performance and Tuning	J. Vatz	TSR	1,1	Feb. 1985	83934
Understanding PATHWAY Statistics	R. Wong	TJ	2,2	Spring 1984	83932
POET					
Designing Client/Server Applications for OLTP on Guardian 90 Systems	W. Culman	TSR	8,3	Fall 1992	89803
PS MAIL					
Enhancements to PS MAIL	R. Funk	TSR	3,1	March 1987	83939

Article title	Author(s)	Publication	Volume, Issue	Publication date	Part number
RDF					
RDF: An Overview	J. Guerrero	TSR	7,2	Oct. 1991	65248
RDF Synchronization	F. Jongma, W. Senf	TSR	8,2	Summer 1992	69848
RESPOND					
The RESPOND OLTP Business Management System for Manufacturing	H. Bolling, W. Bronson	TSR	9,1	Winter 1993	89804
RSC					
Implementing Client/Server Using RSC	M. Iem, T. Kocher	TSR	8,3	Fall 1992	89803
SAFEGUARD					
Dial-In Security Considerations	P. Grainger	TSR	7,2	Oct. 1991	65248
Distributed Protection with SAFEGUARD	T. Chou	TSR	2,2	June 1986	83937
Enhancing System Security With Safeguard	C. Gaydos	TSR	7,1	April 1991	46988
SNAX					
An Overview of SNAX/CDF	M. Turner	TSR	5,2	Sept. 1989	28152
A SNAX Passthrough Tutorial	D. Kirk	TJ	2,2	Spring 1984	83932
SNAX/APC: Tandem's New SNA Software for Distributed Processing	B. Grantham	TSR	3,1	March 1987	83939
SNAX/HLS: An Overview	S. Saltwick	TSR	1,2	June 1985	83935
SPOOLER					
Sizing the Spooler Collector Data File	H. Norman	TSR	4,1	Feb. 1988	11078
TACL					
Debugging TACL Code	L. Palmer	TSR	4,2	July 1988	13693
TACL, Tandem's New Extensible Command Language	J. Campbell, R. Glascock	TSR	2,1	Feb. 1986	83936
TAL					
New TAL Features	C. Lu, J. Murayama	TSR	2,2	June 1986	83837
TCM					
Capacity Planning With TCM	W. Highleyman	TSR	7,2	Oct. 1991	65248
TLAM					
TLAM: A Connectivity Option for Expand	K. MacKenzie	TSR	7,1	April 1991	46988
TMDS					
C00 TMDS Performance	J. Mead	TSR	4,1	Feb. 1988	11078
Enhancements to TMDS	L. White	TSR	3,2	Aug. 1987	83940
Introducing TMDS, Tandem's New On-line Diagnostic System	J. Troisi	TSR	1,2	June 1985	83935
TMF					
Improvements in TMF	T. Lemberger	TSR	1,2	June 1985	83935
TMF and the Multi-Threaded Requester	T. Lemberger	TJ	1,1	Fall 1983	83930
TMF Autorollback: A New Recovery Feature	M. Pong	TSR	1,1	Feb. 1985	83934
TNS/R					
Debugging Accelerated Programs on TNS/R Systems	D. Cressler	TSR	8,1	Spring 1992	65250
Improving Performance on TNS/R Systems With the Accelerator	M. Blanchet	TSR	8,1	Spring 1992	65250
Overview of Tandem NonStop Series/RISC Systems	L. Faby, R. Mateosian	TSR	8,1	Spring 1992	65250

Article title	Author(s)	Publication	Volume, Issue	Publication date	Part number
TRANSFER					
The TRANSFER Delivery System for Distributed Applications	S. Van Pelt	TJ	2,2	Spring 1984	83932
TXP					
The High-Performance NonStop TXP Processor	W. Bartlett, T. Houy, D. Meyer	TJ	2,1	Winter 1984	83931
The NonStop TXP Processor: A Powerful Design for On-line Transaction Processing	P. Oleinick	TJ	2,3	Summer 1984	83933
V8					
The V8 Disc Storage Facility: Setting a New Standard for On-line Disc Storage	M. Whiteman	TSR	1,2	June 1985	83935
VIEWSYS					
VIEWSYS: An On-line System-resource Monitor	D. Montgomery	TSR	1,2	June 1985	83935
VLX					
NonStop VLX Hardware Design	M. Brown	TSR	2,3	Dec. 1986	83938
NonStop VLX Performance	J. Enright	TSR	2,3	Dec. 1986	83938
The VLX: A Design for Serviceability	J. Allen, R. Boyle	TSR	3,1	March 1987	83939
XL8					
Data-encoding Technology Used in the XL8 Storage Facility	D. S. Ng	TSR	2,2	June 1986	83937
Plated Media Technology Used in the XL8 Storage Facility	D. S. Ng	TSR	2,2	June 1986	83937

TandemSystemsReview*OrderForm*

Use this form to order new subscriptions, change subscription information, and order back issues.

☐ I am a Tandem customer. My Tandem sales representative is _____.

☐ I am not a Tandem customer and am enclosing a check or money order for the requests indicated on this form. (Subscriptions are $75 per year and each back issue is $20. Make checks payable to Tandem Computers Incorporated.)

Subscription Information

☐ New subscription

☐ Update to subscription information
Subscription number: _____

Your subscription number is in the upper right corner of the mailing label.

_____
COMPANY

_____
NAME

_____
JOB TITLE

_____
DIVISION

_____
ADDRESS

_____

_____
COUNTRY

_____
TELEPHONE NUMBER (include all codes for U.S. dialing)

Title or position:

☐ President/CEO
☐ Director/VP information services
☐ MIS/DP manager
☐ Software development manager
☐ Programmer/analyst
☐ System operator
☐ End user
☐ Other: _____

Your association with Tandem:

☐ Tandem customer
☐ Third-party vendor
☐ Consultant
☐ Other: _____

Back Issue Requests

Number of copies

Tandem Systems Review

_____ Vol. 1, No. 1, Feb. 1985
_____ Vol. 1, No. 2, June 1985
_____ Vol. 2, No. 1, Feb. 1986
_____ Vol. 2, No. 2, June 1986
_____ Vol. 2, No. 3, Dec. 1986
_____ Vol. 3, No. 1, March 1987
_____ Vol. 3, No. 2, Aug. 1987
_____ Vol. 4, No. 1, Feb. 1988
_____ Vol. 4, No. 2, July 1988
_____ Vol. 4, No. 3, Oct. 1988
_____ Vol. 5, No. 1, April 1989
_____ Vol. 5, No. 2, Sept. 1989
_____ Vol. 6, No. 1, March 1990
_____ Vol. 6, No. 2, Oct. 1990
_____ Vol. 7, No. 1, April 1991
_____ Vol. 7, No. 2, Oct. 1991
_____ Vol. 8, No. 1, Spring 1992
_____ Vol. 8, No. 2, Summer 1992
_____ Vol. 8, No. 3, Fall 1992
_____ Vol. 9, No. 1, Winter 1993
_____ Vol. 9, No. 2, Spring 1993
_____ Vol. 9, No. 3, Summer 1993
_____ Vol. 9, No. 4, Fall 1993
_____ Vol. 10, No. 1, Jan. 1994

Tandem Journal

_____ Vol. 1, No. 1, Fall 1983
_____ Vol. 2, No. 1, Winter 1984
_____ Vol. 2, No. 2, Spring 1984
_____ Vol. 2, No. 3, Summer 1984

For questions or ordering information, call 800-473-5868 in the U.S. and Canada or +1-408-285-0665 in other countries.

Send this form to:

Tandem Computers Incorporated
Tandem Systems Review, Loc 208-65
10400 Ridgeview Court
Cupertino, CA 95014-0723
FAX: +1-408-285-0840

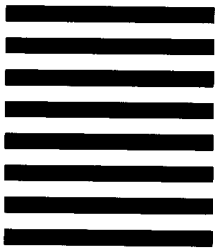
Tandem employees must order their subscriptions and back issues through Courier.

Menu sequence: Marketing Information → Literature Orders → Technical Marketing Pubs (TSR)

▲ FOLD

▲ FOLD

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES



BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 482 CUPERTINO, CA U.S.A.

POSTAGE WILL BE PAID BY ADDRESSEE

TANDEM SYSTEMS REVIEW
LOC 208-65
TANDEM COMPUTERS INCORPORATED
19333 VALLCO PARKWAY
CUPERTINO, CA 95014–9862

▼ FOLD

▼ FOLD

TandemSystemsReview*ReaderSurvey*

The purpose of this questionnaire is to help the *Tandem Systems Review* staff select topics for publication. Postage is prepaid when mailed in the United States. Readers outside the U.S. should send their replies to their nearest Tandem sales office.

1. How useful is each article in this issue?

Product Update

01 ☐ Indispensable 02 ☐ Very 03 ☐ Somewhat 04 ☐ Not at all

A Hardware Overview of the NonStop Himalaya K10000 Server

05 ☐ Indispensable 06 ☐ Very 07 ☐ Somewhat 08 ☐ Not at all

Extending the Client/Server Model With Object-Oriented Technology

09 ☐ Indispensable 10 ☐ Very 11 ☐ Somewhat 12 ☐ Not at all

Basic Uses and New Features of Extended GDS

13 ☐ Indispensable 14 ☐ Very 15 ☐ Somewhat 16 ☐ Not at all

Technical Information and Education

17 ☐ Indispensable 18 ☐ Very 19 ☐ Somewhat 20 ☐ Not at all

2. I specifically would like to see more articles on (select one):

21 ☐ Overview discussions of new products and enhancements
22 ☐ Performance and tuning information
23 ☐ High-level overviews on Tandem's approach to solutions
24 ☐ Application design and customer profiles
25 ☐ Technical discussions of product internals
26 ☐ Strategic information and statements of direction
27 ☐ Other _____

3. Your title or position:

28 ☐ President, VP, Director 29 ☐ Systems analyst 30 ☐ System operator
31 ☐ MIS manager 32 ☐ Software developer 33 ☐ End user
34 ☐ Other _____

4. Your association with Tandem:

35 ☐ Tandem customer 36 ☐ Tandem employee 37 ☐ Third-party vendor 38 ☐ Consultant
39 ☐ Other _____

5. Comments

_____

_____

_____

_____

NAME

_____

COMPANY NAME

_____

ADDRESS

_____

_____

1/94

▲ FOLD

▲ FOLD

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 482 CUPERTINO, CA U.S.A.

POSTAGE WILL BE PAID BY ADDRESSEE

TANDEM SYSTEMS REVIEW
LOC 208-65
TANDEM COMPUTERS INCORPORATED
19333 VALLCO PARKWAY
CUPERTINO, CA 95014–9862

▼ FOLD

▼ FOLD

TANDEM

Tandem Computers Incorporated
19333 Vallco Parkway
Cupertino, CA 95014-2599

MARC BRANDIFINO
LOC NUM 56-00
NEW YORK NY DOWNTOWN DISTRICT

Part No. 104396

1/94 Printed in USA