Solutions Design Guide NonStop® Software for Windows NT® Server Clusters

Abstract

This guide contains high-level information about developing NonStop Software-based solutions that will run on Windows NT Server (version 4.0 and higher) systems.

Product Version

NonStop Software for Windows NT Server

Supported Releases

N.A.

Part Number	Published
133208	February 1998

Document History

Part Number	Product Version	Published
133208	NA	February 1998

New editions incorporate any updates issued since the previous edition.

A plus sign (+) after a release ID indicates that this manual describes function added to the base release, either by an interim product modification (IPM) or by a new product version on a .99 site update tape (SUT).

Ordering Information

For manual ordering information: domestic U.S. customers, call 1-800-243-6886; international customers, contact your local sales representative.

Document Disclaimer

Information contained in a manual is subject to change without notice. Please check with your authorized Tandem representative to make sure you have the most recent information.

Export Statement

Export of the information contained in this manual may require authorization from the U.S. Department of Commerce.

Examples

Examples and sample programs are for illustration only and may not be suited for your particular purpose. Tandem does not warrant, guarantee, or make any representations regarding the use or the results of the use of any examples or sample programs in any documentation. You should verify the applicability of any example or sample program before placing the software into productive use.

U.S. Government Customers

FOR U.S. GOVERNMENT CUSTOMERS REGARDING THIS DOCUMENTATION AND THE ASSOCIATED SOFTWARE:

These notices shall be marked on any reproduction of this data, in whole or in part.

NOTICE: Notwithstanding any other lease or license that may pertain to, or accompany the delivery of, this computer software, the rights of the Government regarding its use, reproduction and disclosure are as set forth in Section 52.227-19 of the FARS Computer Software—Restricted Rights clause.

RESTRICTED RIGHTS NOTICE: Use, duplication, or disclosure by the Government is subject to the restrictions as set forth in subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software clause at DFARS 52.227-7013.

RESTRICTED RIGHTS LEGEND: Use, duplication or disclosure by the Government is subject to restrictions as set forth in paragraph (b)(3)(B) of the rights in Technical Data and Computer Software clause in DAR 7-104.9(a). This computer software is submitted with “restricted rights.” Use, duplication or disclosure is subject to the restrictions as set forth in NASA FAR SUP 18-52 227-79 (April 1985) “Commercial Computer Software—Restricted Rights (April 1985).” If the contract contains the Clause at 18-52 227-74 “Rights in Data General” then the “Alternate III” clause applies.

U.S. Government Users Restricted Rights — Use, duplication or disclosure restricted by GSA ADP Schedule Contract.

Mnemonics and Acronyms

OSS is a mnemonic for Open System Services. DCE is a mnemonic for Distributed Computing Environment.

Trademarks and Service Marks

Copyrights

Disclaimer

License Agreement

Solutions Design Guide NonStop® Software for Windows NT® Server Clusters

Examples

Figures

Tables

About the Solutions Design Guide

Tandem® NonStop Software is a family of platform-independent software designed to run on both Tandem NonStop Himalaya® servers and on Microsoft® Windows NT Server systems based on the Intel microprocessor. The *Solutions Design Guide* provides a wide range of high-level information about developing enterprise-level business solutions: in particular, highly available and scalable distributed online transaction processing (OLTP) and decision support systems (DSS) targeted to be built with key software products from the NonStop Software for Windows NT Server product line.

What the Guide Is About

The *Solutions Design Guide* is a high-level overview about developing online transaction processing (OLTP) applications, decision support systems (DSS), and interactive transaction processing applications on Windows NT Server 4.x using the following software products from the NonStop Software for Windows NT Server product line:

- NonStop SQL/MX, an ISO/ANSI-92 compliant SQL database
- NonStop TUXEDO for Windows NT Server, an XATMI-compliant transaction-processing (TP) monitor

The applications developed using these products will run only on Intel-based Windows NT Server systems. The business solutions discussed in the guide include:

- Services (server-based applications) that implement NonStop TUXEDO for Windows NT Server software and NonStop SQL/MX on the Windows NT Server platform
- Client applications that run on the Windows client operating systems (Windows NT Workstation and Windows 95) and connect to server applications running on Windows NT Server systems

What This Guide Doesn't Cover

This guide is for use with NonStop Software for Windows NT Server (hereafter, referred to simply as "NonStop Software"). The product line contains software for building business solutions, not system-level software. If you wish to develop system-level software that will operate with NonStop Software at a lower level — utilities, e-mail gateways, system-monitoring software, and the like — please look for more information on Tandem's NonStop Software Web site (http://www.nonstopsw.com). You can also ask your Tandem representative for information, or call 1-800-4TANDEM (1-800-482-6336) and request any information you may need.

If you wish to develop system-level software for Windows NT Server, refer to any of the numerous publications that exist for this purpose. Microsoft's Web site contains technical information, software developer kits, and other resources for developers.

Who Should Use This Guide

This guide is for Windows NT Server system business application developers. The audience may include independent software vendors (ISVs) who provide applications, such as general ledger accounting and other business applications that use an underlying database, transaction services, and transaction management facilities. The audience may also include corporate in-house developers who are designing custom line-of-business applications within their companies.

How This Guide is Organized

This *Solutions Design Guide* contains five major topic sections, two appendices, and a glossary.

Section 1, Introduction provides an overview of Tandem's NonStop Software product line and summarizes the product line's key benefits for both Windows NT Server application developers and Tandem Himalaya server developers.

Section 2, Design Considerations, provides a high level overview of the three-tier client/server architecture and what developers should consider before they begin designing solutions.

Section 3, Getting Started, provides an overview of application development for Windows NT Server system clusters, with high-level recommendations for using NonStop Software.

Section 4, Designing Portable Solutions, discusses relevant issues for developing applications for Windows NT Server so that they can be easily ported to other platforms, including the Tandem NonStop Himalaya server.

Section 5, NonStop SQL/MX Solutions, provides information for developing solutions using NonStop SQL/MX for Windows NT Server. The section includes an overview of the product features, functions, and components; recommendations about various design approaches for decision support systems; Internet-related technologies, such as httpd (Web) servers; and an API reference that includes relevant information about ISO/ANSI-92 SQL and ODBC.

Section 6, NonStop TUXEDO Solutions, provides information for developing solutions using NonStop TUXEDO for Windows NT Server. The section includes an overview of the product features, functions, and components; recommendations about various design approaches for OLTP, including design alternatives in the context of various Internet-related technologies; and an API reference that includes relevant information about the TUXEDO APIs.

Appendix A, Windows NT® Server Cluster Notes discusses the key architectural differences between different processor architecture models (uniprocessor, symmetric multiprocessor, and massively parallel processor) and provides an overview of clustering in the Microsoft® Windows NT Server environment, the Microsoft Cluster Server API, and related concepts. This section also provides an brief overview of the Windows NT Server operating system architecture and Tandem NonStop Kernel operating system server architecture within the context of clustering.

Appendix B, Enterprise Computing Primer provides a very basic enterprise computing primer, including brief discussion of transaction processing, the ACID concept and two-phase commit protocol for distributed online transaction processing.

The Glossary defines acronyms, abbreviations, and some of the product names referred to in this guide.

Where to Go for More Information

The *Solutions Design Guide* provides a high-level overview of applications development for the NonStop Software product set on Windows NT Server systems. For more in-depth information about any topics contained in this guide, please see the appropriate publications, available on the CD ROM, from among those in the table below.

Table i. NonStop Software Product Technical Publications

NonStop SQL/MX	**NonStop TUXEDO for Windows NT Server**
NonStop SQL/MX Reference Manual	*NonStop TUXEDO for Windows NT Server Reference Manual*
NonStop SQL/MX Programming Manual for C and COBOL	*NonStop TUXEDO for Windows NT Server Programmer's Guide*
NonStop SQL/MX Installation Guide	*NonStop TUXEDO for Windows NT Server Installation Guide*
NonStop ODBC/MX for Windows NT Server	

For an overview of Tandem NonStop Himalaya server architecture and the NonStop online transaction processing environment, see *Introduction to NonStop Transaction Processing*. Many Tandem publications are available from the Tandem Web site (http://www.tandem.com). You may also call Tandem directly at 1-800-4TANDEM (1-800-482-6336) and request any publication.

For an overview of online transaction processing, see:

- *Principles of Transaction Processing for the Systems Professional*, by Philip A. Bernstein and Eric Newcomer. This 1997 publication from Morgan Kaufman Publishing, Inc., contains up-to-date information about TP monitors (BEA TUXEDO, IBM/Encina Transarc, AT&T Top End, the IBM CICS™[1] API, Tandem Pathway, and Microsoft Transaction Server) and provides a conceptual overview of the TP architecture. (ISBN 1-55860-415-4)
- *Transaction Processing: Concepts and Techniques* by Jim Gray and Andreas Reuter. Morgan Kaufman Publishing, Inc. 1994. (ISBN 1-55860-190-2)

1. CICS is a trademark of IBM Corporation.

For detailed comprehensive information about online transaction processing in the TUXEDO environment, see *Building Client/Server Applications using TUXEDO* by Carl Hall. John Wiley Sons, 1996.

For information about the theory of database design, see *An Introduction to Database Systems, Sixth Edition*, by C.J. Date. Addison-Wesley. 1995. (ISBN 0-201-54329-X) Also see *Understanding the New SQL: A Complete Guide* by Jim Melton and Alan Simon. Morgan Kaufman Publishing, Inc. 1996. (ISBN 1-55860-245-3)

For a high-level overview of distributed computing, see *Distributed Computing: Implementation and Management Strategies*, edited by Raman Khanna. PTR Prentice Hall. (ISBN 0-13-220138-0).

Client/server Architecture. (2nd Edition, 1996; McGraw Hill) by Alex Berson. (ISBN 0-07-005664-1)

For more information about Windows NT Server, see:

- *Inside Windows NT*, by Helen Custer. Microsoft Press. 1995. (ISBN 1-55615-481-X)
- *Advanced Windows: Third Edition (The Developers Guide to the Win32® API for Windows NT 4.0 and Windows 95)*, by Jeffrey Richter. Microsoft Press. 1997. (ISBN 1-57231-548-2)

For more information about the NonStop Software product line, please contact your Tandem representative, or see the NonStop Software Web site (http://www.nonstopsw.com).

Your Comments Invited

If you have any questions, comments, or suggestions about the content of this *Solutions Design Guide*, please send an Internet mail message to pubs_comments@tandem.com. Be sure to include your name, company name, address, and phone number in your message, and please include the part number and title of this guide.

1 Introduction

High availability and scalability are two of the most important requirements for enterprise-wide, business-critical applications. And providing software and hardware to meet these requirements has been Tandem's strength for more than two decades. With the NonStop Software product line, Tandem is now bringing that strength to the Windows NT Server system environment. NonStop Software for Windows NT Server brings high availability, scalability, and manageability to the Windows NT Server platform and will enable developers to provide these benefits for the first time on applications developed for Windows NT Server systems: specifically, Windows NT Server system *clusters* — multiple Windows NT Server systems connected together to form a single logical server system.

Clusters and clustering technology for Windows NT Server systems have been widely covered in the trade and business press during the past year, and for good reason: For Windows NT Server to be a viable enterprise application server platform, it must be able to achieve better scalability than that enabled by SMP (symmetric multiprocessor) architecture alone. Microsoft has turned to vendors, including Tandem, that have a history of clustering expertise on other platforms, and has been working with such vendors to bring clustering technology to the Windows NT Server platform.

Tandem NonStop Software for Windows NT Server products are designed to run on Windows NT Server system clusters.[1] The product line for Windows NT Server systems includes the NonStop SQL/MX for Windows NT Server database and NonStop TUXEDO for Windows NT Server transaction processing (TP) monitor. These products will enable developers to create highly available and scalable business applications for enterprise-level online transaction processing (OLTP) and decision support systems (DSS) that will run on Windows NT Server clusters.

But, most importantly, NonStop Software will enable developers to create highly available and scalable applications without special programming, without coding to Microsoft's underlying clustering API, and without building infrastructure. This means you'll get high availability and scalability on your Windows NT Server solution at a lower development cost, because the fundamental software components required for business-critical computing — much of the so-called "plumbing" or infrastructure — is included as part of the NonStop Software for Windows NT Server product set (hereafter, referred to simply as "NonStop Software"). Better still, the product set and infrastructure is transparently integrated with the Windows NT Server operating system, in true Microsoft BackOffice application style.

This guide discusses developing NonStop Software solutions for the Windows NT Server platform. This section introduces the NonStop Software product line and some of its advantages for developers.

1. NonStop Software products are also available for Tandem Himalaya servers. The complete NonStop Software product line will include versions that run on Windows NT Server clusters and Tandem Himalaya servers.

Advantages of Developing NonStop Software Solutions

NonStop Software has numerous advantages for developers, business users, and administrators, including:

- Exploit Windows NT Server clusters — without extra coding
- Achieve highly available and scalable applications on Windows NT Server clusters
- Leverage your development efforts on multiple platforms
- Achieve greater market opportunity for your application
- Use industry-standard APIs to write a NonStop Software-based application
- Gain the ease-of-use of Tandem's single application image
- Develop applications compliant with the Microsoft BackOffice applications suite using NonStop Software

Each of these advantages is discussed in the following subsections.

Exploit Windows NT Server Clusters — Without Extra Coding

Applications developed for the NonStop SQL/MX database and NonStop TUXEDO will ***automatically exploit Windows NT Server clusters***. No special coding is necessary, nor is awareness (on the part of the developer) of the underlying clustering API or interconnect mechanism and related cluster-enabling technologies.

Achieve Highly Available and Scalable Applications on Windows NT Server Clusters

You can achieve highly available and scalable applications on a relatively low-cost platform, and be assured that the Tandem fundamentals are provided by infrastructure components such as Tandem's industry-leading transaction management facility and transaction services. NonStop Software will scale across a 16-node cluster of four-way Windows NT Server nodes.

Leverage Your Development Efforts on Multiple Platforms

The Windows NT Server system as an applications server has a growing presence in the enterprise marketplace. Tandem developers can leverage their mission-critical application development expertise into a broader target market; Windows NT Server developers can do the same: You can develop applications for Tandem Himalaya servers, or for clustered Windows NT Server systems.

Because the NonStop SQL/MX database is ISO/ANSI-92 SQL compliant and because the NonStop TUXEDO transaction processing monitor uses the de facto industry-standard ATMI interface, you can essentially develop your application once and deploy it on TUXEDO implementations from various vendors, across heterogeneous platforms. Depending upon your long-term strategy, you will have to keep certain

recommendations in mind. See Section 4, Designing Portable Solutions, for a summary of recommendations for multiplatform developers.

Achieve Greater Market Opportunity for Your Application

Tandem NonStop Himalaya developers can extend their applications into new markets by providing new functionality on Windows NT Server systems where it makes sense to do so: for example, by implementing branch or satellite location functions on Windows NT Server systems and leaving back-end or central processing capabilities on Tandem Himalaya servers.

Windows NT developers can build applications on uniprocessor Windows NT Server systems with the assurance that these applications can be scaled up if need be, either through combining multiple Windows NT nodes into clusters or by recompiling the application for Tandem's NonStop Kernel on a Tandem Himalaya server.

Use Industry-standard APIs to Write a NonStop Software-based Application

Developers can create solutions implementing NonStop Software using industry-standard APIs: there is no "NonStop API." Developers building solutions around NonStop SQL/MX will use ANSI/ISO-92 SQL or ODBC; developers building solutions around NonStop TUXEDO will use the industry-standard ATMI (Application to Transaction Manager Interface).

Gain the Ease-of-Use of Tandem's Single Application Image

A single application image results in a more manageable application for both operations staff and end-users. Tandem NonStop Himalaya systems provide this level of manageability, and it's one of the added values that Tandem is bringing to the Windows NT Server cluster marketplace.

Develop Microsoft BackOffice Compliant Applications with NonStop Software

Because both NonStop SQL/MX database and NonStop TUXEDO meet the requirements for Microsoft BackOffice logo compliance, it's likely that you'll be successful making your own Microsoft BackOffice compliant application.

BackOffice compliance means that the application installs using the standard Windows NT installation tools, that events from the application are captured in the Windows NT Server event log, and that configuration information is stored in the Windows NT Server Registry. In addition, there's an OLE Automation interface to the NonStop Software product GUI, which provides windows or dialogs for rudimentary SQL commands, utility operations, configuring the database, and monitoring NonStop Software operations.

Types of Applications You Can Develop

NonStop Software is designed to meet requirements for two strategic areas for application developers: transaction processing and decision support. This publication is for developers creating these types of business applications, rather than system-level software. The product set includes the NonStop SQL/MX database and NonStop TUXEDO.

With the NonStop SQL/MX database, developers can create both decision support systems and online transaction processing applications to run on Windows NT Server clusters. NonStop SQL/MX is a high-performance, distributed relational database management system conforming to the ISO/ANSI-92 SQL standard. In addition, NonStop SQL/MX complies with Microsoft BackOffice logo criteria.

NonStop SQL/MX will provide linear scalability from one SMP node to a cluster of 16 SMP nodes. When combined with the load-balancing and scalability features in NonStop TUXEDO, OLTP applications built using NonStop SQL/MX will be able to handle a transaction volume equivalent to or greater than that handled by the most powerful UNIX SMP systems, at a significantly lower cost. NonStop TUXEDO provides many features developers need for distributed online transaction processing applications, including support for client/server communications and transaction protection. For additional details, including an overview of supported APIs, see Section 5, NonStop SQL/MX Solutions.

Tandem's NonStop TUXEDO complies with the industry-leading BEA Systems, Inc., TUXEDO product and underlying ATMI (Application-Transaction Manager Interface). BEA Systems' TUXEDO is considered the highest performance open OLTP monitor and is used extensively in TPC benchmarks by many hardware and software vendors. For additional details, including an overview of supported APIs, see Section 6, NonStop TUXEDO Solutions.

NonStop Software Provides Scalable Windows NT Server Clusters — Today

Clustering technology is key to Windows NT Server's adoption as an enterprise application platform. That's why Microsoft has been working with Tandem (and other vendors) since May 1996 to bring clustering technology to the Windows NT Server platform. The first generation of Microsoft's clustering technology will be incorporated in a new version of Windows NT Server — Windows NT Server, Enterprise Edition, or Windows NT Server/E — which Microsoft has announced it will release during the summer of 1997. Windows NT Server/E will incorporate the first phase of Microsoft's clustering APIs and other features for enterprise-class computing.

The clustering component is the Microsoft Cluster Server (MSCS) APIs. Formerly known as the "Wolfpack" APIs, MSCS is being released by Microsoft in two carefully controlled phases: MSCS Phase 1 will provide 2-node fail over, which will enhance availability but does nothing to enhance scalability. This is the version that will be released by Microsoft when it ships Windows NT Server/E later this summer. MSCS Phase 2, which Microsoft has stated will enter beta test in 1998, will enable more than two Windows NT Server systems to be connected together for higher performance and

scalability. (For a high-level overview of the MSCS APIs, see Microsoft Cluster Server and Windows NT Server Clusters on page A-5.)

Nonetheless, you can begin developing solutions today for the Windows NT Server cluster platform, using Tandem NonStop Software. Because the 2-node fail over type of clustering is not considered a true enterprise-class application platform, Tandem has developed key clustering infrastructure as needed to support the initial releases of NonStop Software, so that developers can begin designing and prototyping solutions on n-node Windows NT Server clusters — today — which can be tested and deployed on production-ready NonStop Software when it ships later this year.

Not only is this infrastructure seamlessly integrated with the Windows NT Server architecture, but it enables the NonStop SQL/MX and NonStop TUXEDO software products to automatically exploit all the Windows NT Server systems that comprise the cluster.

Thus, while the Microsoft Windows NT Server/E will enable greater availability of Windows NT Server applications, the Tandem NonStop Software will enable scalability.

Note: For more information about Windows NT Server clusters; the MSCS APIs; and Microsoft Windows NT Server, Enterprise Edition, see Microsoft's Web site (http://www.microsoft.com/ntserver/info).

2 Design Considerations

The first step in any application design is to identify the business requirements. To some extent any application designed to use NonStop Software products will have high availability and scalability as basic requirements. A discussion of how to identify business requirements is beyond the scope of this document, but the better you define the business needs and goals, the easier it will be to specify your technical requirements.

Regardless of the specific technical requirements for your solution, application designers should always consider issues such as application architecture, instrumentation, error handling, internationalization, and security. This section includes a high-level discussion of these issues.

Before You Begin

Developers reading this guide will no doubt have numerous and wide-ranging experiences, including but not limited to development of:

- Standalone applications for uniprocessor Windows NT Server systems
- Client/server applications for uniprocessor Windows NT Server systems
- Client/server applications for SMP Windows NT Server systems
- Client/server applications for UNIX system servers, including SMP and clustered UNIX systems
- Tandem NonStop Himalaya server applications (NonStop TS/MP, NonStop TM/MP, and NonStop TUXEDO, using a NonStop SQL/MP or an Enscribe database)

Whatever your perspective as a developer, it's important to question assumptions based on prior experience. For example, using threads on a uniprocessor will have different effects than using threads on an SMP model system, and different implications still in a cluster. There's no need to use threads with NonStop SQL/MX, because the database engine itself is multithreaded and will spread the processing load across all processors that comprise the system.

Another example of an instance in which one should question assumptions is in the use of shared memory: in uniprocessor and SMP environments, you can use shared memory safely for interprocess communications, but as a general rule, shared memory should be avoided in clustered environments. The use of threads and of shared memory are just two areas that need to be considered when designing your application.

Factors to Consider in Application Design

The client/server architecture has become the preferred model for enterprise class applications. A client/server model splits functionality between a *server* process that provides services to other processes and a *client* process that invokes the services provided by another process. There are many factors to consider in defining the

boundaries of client processes and server processes. Some of these factors are contained in the following lists.

Client Considerations

- Platform type (hardware and operating system)
- APIs and protocols supported on the client workstation (Winsock, RPC, TCP/IP, and so on)
- Base hardware configuration and sizing issues (CPU model, RAM available, quantity and type of local permanent storage)
- IPC (interprocess communication mechanisms) available at client platform
- Theoretical network bandwidth vs. actual throughput during peak network load
- Programming model (thin client)
- Web technology required (browser client, helper applications, and so on)
- Life expectancy of the client application

Server Considerations

- Number of users and potential users
- Location of all users and potential users
- Theoretical network bandwidth vs. available throughput during peak load
- Total number of users that would comprise peak load
- Life expectancy of the application
- Time sensitivity of the application
- Extensibility required of the application over the short, intermediate, and long term
- IPC (interprocess communication mechanisms) available at server, across platforms
- APIs and protocols available on the server platform (Winsock, RPC, TCP/IP, X.25, and so on)
- Number and type of different platforms on which the application must be deployed
- Support for mobile or remote users

Data Considerations

- Quantity of data
- Location of all data
- Logical structure of data
- Physical structure and location of data

- Type of data (relational, object, streaming, and so on)
- Time sensitivity of data (immediate updates required? extracts sufficient? data consistency issues)
- APIs and data access protocols supported

The design of the database itself is another key issue. Should the tables be partitioned across the nodes of the cluster? How should the tables be structured? Relationships among data elements need to be defined and configured. The goals of a transaction processing system and of a decision support system are very different, and place different requirements on database design. Many textbooks and other reference materials cover these topics in full; see Where to Go for More Information on page xiii for specific references.

Many of the considerations listed above relate to physical design issues, but the other key consideration is the logic of the application itself, and how the logic should be partitioned between client processes and server processes. An online business application performs three main types of functions: it interacts with users, it performs business functions (implements business rules), and it accesses a database. These activities can be partitioned in many ways. The two primary approaches for splitting these functions are the two-tier client/server model and the three-tier client/server model.

The two-tier client/server model splits functionality between client and a database-server, while a three-tier model additionally isolates the data and the server application process, leaving just the business rules and application logic at the middle or server application process. Each model is discussed briefly below.

Two-Tier Client/Server Model

In a two-tier design, the bulk of an application's code may be in the client application or the server application. The client may make calls directly to a service or database system, for example, when an ODBC-based decision support client application is querying the RDBMS directly. This design leads to one database connection per user, which may be sufficient for a small number of users, but doesn't scale well as the number of users grows. (Note that ODBC can also be used in the context of three-tier applications. Later in this guide, in Section 5, NonStop SQL/MX Solutions, you'll see how to develop three-tier client/server applications using ODBC within the context of the server application.)

An alternate two-two design might have the business rules contained in the database, as stored procedures and triggers[1]. Stored procedures and triggers are specific to the database implementation, and changing the business rules requires changing the stored procedure (or trigger) logic, as well as potentially changing all client applications that call the procedure.

The two-tier model is appropriate for many workgroup or departmental applications, but it is less extensible and flexible than the three-tier model. When the business rules are

1. A trigger defines an action to take place when a particular database-related event occurs. A procedure is a compiled data access method, stored in the database, that cannot be dynamically altered by end-users; thus, both comprise programming logic which is tightly linked to the database.

tightly linked to the client process and server process from end-to-end, the whole needs to be changed rather than just the appropriate subcomponent.

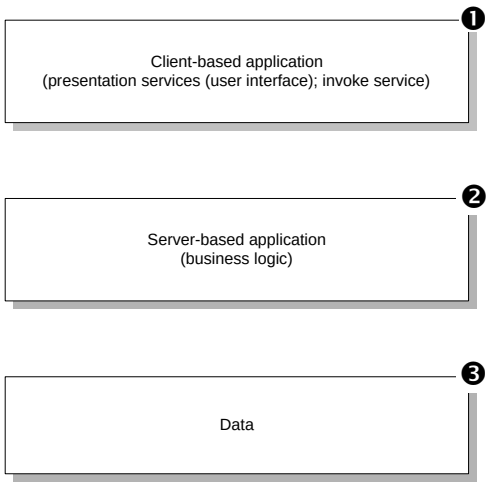
Using stored procedures, for example, ties the database structure and the program logic together, which makes changing one or the other more difficult, and thus, more costly. Managing software upgrades and changes for hundreds or thousands of client application users is a costly endeavor, in addition to being error prone. Because of these and many other issues, Tandem, Microsoft, and other vendors and developers recommend using a three-tier client/server model for distributed applications design.

Three-Tier Client/Server Model

The three-tier client/server model is considered to be easier to maintain, more manageable, and far more scalable than the two-tier architecture. Unlike the two-tier model in which data and processing logic are tightly coupled, the three-tier client/server model separates the business logic from the data. Because the business logic is isolated in a middle tier, its services can access disparate data sources and can provide services to multiple requesters (clients). That is, unlike the two-tier model with its 1:1 ratio between client and data, the three-tier model enables support for more clients than thre are database connections, because its middle tier allows users to access data through shared connections.

In addition to this key advantage, the three-tier model also lowers the costs associated with software maintenance, distribution, and management. For example, when the business rules for an organization change, only the portion of the application containing the business rules need change, leaving the other modules unaffected. Figure 2-1 shows a simple three-tier model.

Figure 2-1. Three-Tier Client/Server Model



The three-tier model separates client presentation, business logic, and data. The tiers (1, 2, and 3) are logical, not physical, concepts. Application programming interfaces and protocols enable one layer to communicate with another.

Note that the client may be a process running on a server, which in turn may call another server. As this paradigm continues across multiple servers, the three-tier client/server model becomes essentially an n-tier model, with services on one platform calling services on other platforms distributed throughout the enterprise computing environment.

As the concept of three-tier client/server has evolved, a related concept has emerged to identify the quantity of client processing logic in particular: the notion of a “thin” client versus a “fat” client.

“Thin Clients” Compared to “Fat Clients”

There are many different definitions of “thin clients,” ranging from the NC (network computer) platform, with its limited installed operating system and software, to Windows NT Workstations with a full complement of software, but which also use browsers and support functionality downloaded to the client at runtime. To simplify the discussion, this guide defines a thin client as being a logical construct in the context of the three-tier client/server model, in which minimal function is provided directly by the client.

A thin client provides basic connectivity, basic screen display, and data entry facilities, but the bulk of the business function is either provided by the middle tier or downloaded to the client by the middle tier at runtime. The definition is within the context of a given client/server application, not necessarily in the context of the platform on which it runs. For example, a thin client could be a Visual Basic or Visual C++ client application running on Windows 95 desktop workstation that accesses a database server application; or an Internet Explorer (or Netscape Navigator) Web browser client running on a desktop computer (or running on a “network computer”).

“Fat” client typically refers to a client in which the business rules are part of the client application to some extent, or in which processing of data occurs at the client. When business logic is hard-coded into the client application or contained in DLLs (dynamic link libraries) used by the client, changes must be propagated to potentially thousands of end-user workstations, which leads to version control problems and numerous other issues that have become a significant factor in terms of the total cost of ownership (TCO) of desktop and LAN computing. Thin client is becoming the preferred strategy for client development because it potentially relieves a great deal of the application distribution and management burden.

Nonetheless, each client strategy has its own set of advantages and disadvantages and must be explored in the context of the application at hand. The key is to balance the design requirements of the application, the skill of the development staff, and the needs, skill level, and other factors associated with end-users. If you are developing a payroll application to be used in-house by a staff of five accountants and have no plans for expansion beyond these users, a custom-coded Visual Basic front-end using stored procedures or with some of the logic coded into the end-user application may be the most effective solution.

If, on the other hand, you want customers to be able to order products on a secure Web site, then you’ll likely want to build the client portion of the application as a Java applet, which you’ll then host on your external Web site. If the application logic ever needs to change, you have just one piece of code to change, at the Java host site.

Note that a given Java applet implementation may not necessarily be “thin,” particularly if it contains business rules and extensive application logic. Packaging the Java applet with too much function may cause it to become a bottleneck, either by reducing overall network bandwidth or by creating a bottleneck at the server or client process. The key is to find the appropriate division of function in the context of your network computing environment, its distribution of servers and clients, the amount of processing each can support, overall network bandwidth, and all the other factors that go into application design. One development goal should be to transfer as little as possible between client and server, yet still keep the client presentation as current with the state of the server as possible.

Clients, Servers, and State

The discussion of thin and fat clients highlights one of the key issues developers face when designing client/server applications, particularly OLTP applications with distributed transactions: delineating the precise boundaries between client and server, and deciding how each participant will convey information to the other about its condition, or state. The basic Internet and Web communications paradigm is stateless — a client connects to a server, an HTML page is downloaded from the server to the client for display, and there’s no other information exchanged. (This model has been expanded, however, and now there are numerous mechanisms and strategies for saving context in this stateless environment: for example, using “cookies” — a file that gets sent from server to browser at download time or can be read by the server process to convey information.)

Determining the state of each participant at any given point in time (especially when there’s a failure at the client process, at the server process, or between the two of them) is a complex issue and one which developers must consider fully when designing solutions. Every application, with its combination of client and server — Web browsers, Java applets, ActiveX Controls, peer-to-peer communications, RPC communications, wide-area network over slow dial-up links or local-area network, and so on — presents a unique set of challenges.

For in-depth information about the three-tier client/server model in the Windows NT Server applications environment, see the following articles, available from Microsoft Developer Network (http://www.microsoft.com/msdn/):

- “Building Multitiered Client/Server Apps with Visual Basic 4.0, Enterprise Edition”
- “Developing Applications for the Client/Server Model”
- “Leveraging Your Visual C++ Experience on the Internet with Thin Client Technology”

This short list is just a starting point: There are countless articles, tutorials, white papers, technical backgrounders, and application development samples available through the Microsoft Developer Network. Also, see Tandem’s NonStop Software Developers’ Web site for white papers, code samples, and technical briefs for in-depth information about high availability and scalability concepts. (From http://www.nonstopsw.com, select “Developers’ Page.”)

In addition, NonStop TUXEDO, with its conversational interface, supports context-sensitive communications. See *NonStop TUXEDO Programmer's Guide* for additional details and background information.

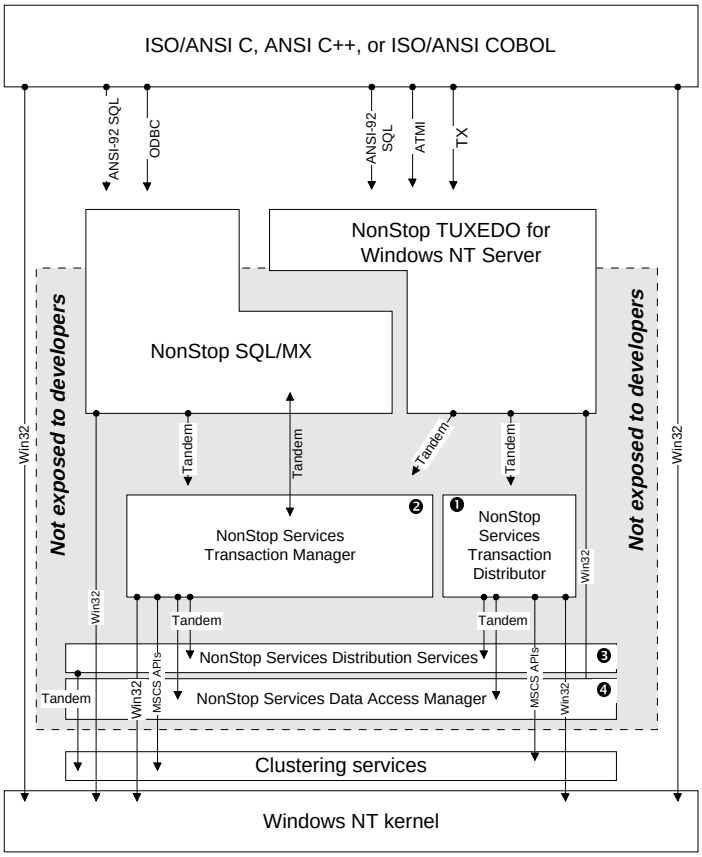
Server-based Application (the Middle Tier)

The three-tier client/server model depicted in Figure 2-1 on page 2-4 is very high level and oversimplified for purposes of discussion. In this section, we'll take a closer look at the middle tier — the "server-based application."

Figure 2-2 is a block diagram of an application architecture that incorporates NonStop SQL/MX and NonStop TUXEDO on the Windows NT Server cluster platform. A well-designed application architecture should be based on open, industry standards so that the application will have a long useful life. "Open" in this context means that applications can be built in such a way that neither the design nor the implementation results in "lock-in" — the situation in which a developer's (or system integrator's) future choices are limited to a particular vendor or technology.

Developers can prevent lock-in by avoiding proprietary standards in favor of open, industry standards, such as those depicted in Figure 2-2. The figure highlights the APIs supported by the NonStop Software product line and how these products fit together in terms of a middle-tier application that hosts services on the Windows NT Server platform. The arrows between components represent calls that might be made to the various APIs supported by a given subsystem or component.

Figure 2-2. Server Application (Middle-Tier) Model



Both NonStop SQL/MX and NonStop TUXEDO are based on open, industry-standard APIs. The underlying NonStop Services layer is not exposed to developers.

The sideways-L-shaped interface between NonStop SQL/MX and NonStop TUXEDO is intended to represent an architecture that encompasses both products in one representation, imparting the idea that a given application may use one or both products. Thus, an application may not incorporate functions of the NonStop TUXEDO product (although in general, Tandem recommends implementing the NonStop TUXEDO transaction processing monitor to enable the greatest scalability). NonStop Software is targeted at developing middle tier applications for the Windows NT Server platform, which access data from NonStop SQL/MX databases.

NonStop SQL/MX and NonStop TUXEDO support industry standards including ISO/ANSI C, ISO/ANSI-92 SQL, ODBC, and ATMI. Not all APIs are shown in the figure; rather, it shows just those that are recommended for use. Depending on which APIs and protocols developers choose, and whether the application makes direct calls to the underlying operating system — for example, using the Win32 API to make direct use of the Windows NT Server operating system — a given solution may be limited to the Windows NT Server system platform, or may be easily ported to other operating systems, including the Tandem NonStop Kernel. (See Designing Portable Solutions on page 4-1, for more details about developing portable solutions.)

This architecture is intended to provide a starting point for developers to begin making design choices. A general guideline is to use ISO/ANSI C and ODBC to NonStop SQL/MX for decision support, or use ISO/ANSI C and NonStop TUXEDO for OLTP applications.

See Near-Term Implementation Issues on page 3-10 for some short-term considerations for developers. See the API Reference sections in Section 5, NonStop SQL/MX Solutions, and Section 6, NonStop TUXEDO Solutions, for more details about SQL and ATMI, respectively.

The figure also provides a view of the relationship between the NonStop Software products and the underlying infrastructure elements, the NonStop Services layer. This layer provides core functionality to the products, but the APIs are not exposed to developers, as discussed in the next section.

Tandem's NonStop Services Layer

Tandem has implemented core components from the NonStop Kernel operating system's underlying clustered infrastructure as needed to support the NonStop Software line. This layer, called NonStop Services, will support n-node clustering on the Windows NT Server platform by the first quarter of 1998. The initial releases of NonStop Software products include this infrastructure.

The NonStop Services infrastructure layer will co-exist with MSCS[1] when it ships, and the Tandem NonStop Software products will take advantage of MSCS where it is appropriate to do so.

For more information about the differences between clusters on Windows NT Server and Tandem NonStop Himalaya systems, see "Making Enterprise-Class Clusters Come Alive" and "NonStop Availability for NonStop Himalaya and Windows NT Server Systems," available on Tandem's Web site (http://www.tandem.com). Some of the issues are discussed briefly in Processing Architectures Affect Scalability and Availability on page A-3.

Application Frameworks

In addition to considering the APIs and protocols used in the client/server application, there are other architectural considerations, including an overall recommendation to develop solutions using an application framework[2].

For graphical user oriented Windows applications, one option is to use the framework provided in the Microsoft Foundation Class (MFC) libraries. You can rapidly prototype and develop your client application using MFC. For example, with Visual C++ and MFC, you can quickly generate the shell of the client application, complete with user

1. Tandem has been working with Microsoft on the Microsoft Cluster Server (MSCS) API (formerly known as "Wolfpack") and clustering for Windows NT Server software since May 1996. In May 1997, Microsoft announced the expected first-phase release of MSCS in a new version of Windows NT Server, the Enterprise edition. Microsoft expects that Windows NT Server/E will support the second phase of its clustering initiative, support for n-node clustering, sometime in 1998. NonStop SQL/MX and NonStop TUXEDO for Windows NT Server will be integrated with the Windows NT Server/E platform when it is ready.

2. Tandem has developed a portability analysis framework, discussed in a Tandem white paper entitled "Future-Proofing Enterprise Applications," available on Tandem's Web site (http://www.tandem.com). Search on "future-proofing" to find the paper.

interface, all resources, header files, and other pieces you need. You supply the programming logic and then compile and link to create an executable file (or DLL, or Java applet, if you're using J++).

From a higher-level perspective, however, an application framework means providing common facilities for handling common, global tasks that might be required of more than one application, particularly when applications are being developed by entire teams of people. For example, common facilities would include code for error handling, including error message text files and other ready-to-deploy modules. Developers should consider the topics that follow — instrumentation, error handling, internationalization, and security — from this perspective.

Instrumentation

Although the NonStop Services layer provides automatic recovery for many low-level subsystems, at a higher level the business application must include instrumentation — programming logic and other software-based mechanisms that identify, capture, or measure changes that occur while an application is operational. For example, your application must include code which defines events that must be recorded to the Windows NT event log.

To write to the Windows NT log file, you can use the ReportEvent function. This function takes a message ID and refers to the message table, which you must provide, to look up the message. Basically, you must create a text file containing all descriptions of all messages your application will generate and log. You then compile this file using message compiler (mc.exe), part of the Win32 SDK. The compiler generates three files: a binary file which is used as a resource in the application, and two files which are included in the source code prior to compilation.

Applications can also include code for recovering from events: Instrumentation provides the means for anticipating and detecting many application failures, and, properly implemented, can also help ameliorate these problems — ideally without affecting end users.

In addition to defining events and how to respond to them, instrumentation can provide information which can be used to improve overall application performance, reliability, and availability. Tandem recommends that you apply instrumentation to all critical modules in your application.

Error Handling

In addition to generating and handling events, you must design the application to handle and recover from errors. The degree of complexity associated with this task will vary, depending upon the communications style of the application and the type of potential errors.

For example, in an RPC (remote procedure call) based application, the back-and-forth, call-response (also called "request/reply") communication stream lends itself to error handling throughout the application. The called procedure sends either a return message or an exception so that the caller can respond accordingly. On the other hand, a peer-to-peer communications style, in which either participant can start and end communications, is more difficult to design, in terms of coding logic for error handling.

As discussed in Clients, Servers, and State on page 2-6, one of the key issues in designing the application is trying to ascertain the state of client and server when there's a failure. Basically, errors fall into two different categories: those that may affect session context, and those that don't affect session context. Errors that don't affect session context are far easier to handle than errors that do affect session context.

For example, in some cases a server hard-disk drive may cause the server to fail over to another hardware component without adversely affecting any of the in-process activities between the client and server (other than a slight delay). In this case, the hardware error doesn't affect session context, and the application may simply need to prompt users to re-submit their last request.

On the other hand, when errors occur between client and server at the session level, some of the context of the interaction may be at the client, and some of it may be at the server. The question becomes how to rebuild the session with the right context information. Sending state information back and forth over the network may reduce overall throughput, so saving context locally may be appropriate.

In any case, developers must still provide application logic to recover from errors, including figuring out how to determine the context. Developers must consider what they can do in the application to enable users to determine whether a transaction committed successfully, even if the response did not reach users' monitors. For example, you might provide end-users with a means of querying the current state of the database in order to determine current context and the amount of work that may need to be re-submitted. However, this may not be an appropriate alternative in every case.

There are no simple answers for these types of development issues, but they are a major concern for solutions designers. In general, what can be said is that the application at a minimum should:

- Retry timouts and open invalidation errors
- Handle breaks in a session, saving context when necessary and enabling restored sessions to accurately continue processing
- Prompt the user to reconnect where appropriate, or, if you design the application to attempt reconnection automatically, the application should alert users to this fact, indicate any required actions, and notify users of the anticipated recovery time
- Provide consistent error messages
- Use a table or text file as input for error messages to enable easy updates and changes and support for different languages
- Send messages to the log file for examination and analysis

Note that the ATMI API of the NonStop TUXEDO product includes APIs for unrequested notification. These function calls can be inserted at various error-checking points in your server application and used to send error messages. The ATMI API requires client code to support receipt of unsolicited messages.

Internationalization

Internationalization[1] refers to the concept of designing software or hardware in such a way as to accommodate local or regional differences in alphabets, languages, currencies, numbers, date-and-time notations, and cultural concepts. Developers can design their solutions for use by people around the world by following some basic guidelines. For example, any information displayed to users (messages, error messages, icons, bitmaps, and the like) should be contained in separate resource files (or other code modules appropriate for the application environment) that can be easily replaced with versions that match the local language.

Tools.h++ from Rogue Wave is a widely available C class library supported on many operating systems, including Windows NT Server and the Tandem NonStop Kernel. Tools.h++ provides facilities (the RWLocale and RWZone classes) to deal with many of these issues at the system level:

- **Alphabets.** Applications must be able to display different alphabets. Windows NT Server uses the Unicode standard for encoding international character sets. Unicode uses a 16-bit character coding scheme, so it can display 65,536 individual characters; font and formatting information is distinct from the characters themselves. The Win32 API subsystem supports both Unicode and ASCII. Object names, path names, file names, and directory names are represented in Unicode.
    - Both NonStop SQL/MX and NonStop TUXEDO support the Unicode standard for error messages. NonStop SQL/MX will support multiple character sets for data in a future release.
- **Languages**. Applications should be able to display titles, menu choices, and status messages in different languages, so it's best to store such display items in a separate code module or in a message catalog separate from program code, which can then be easily edited or replaced.

  Versions of Windows NT Server are available in numerous international languages, including Chinese, Danish, Dutch, Finnish, French, German, Italian, Japanese, Korean, Norwegian, Portuguese (both Brazilian and Castilian), Russian, Spanish, and Swedish.
- **Currencies**. Financial applications should handle differences in currency units and notations. In some cases, an application might need to display values in the notations customary to both vendor and customer.
- **Time and Date.** Time zone and calendar differences must be supported by the application, including variances in names for months, and days of the week, and order of presentation. Time-zone conventions and daylight savings time must also be supported.

Security

One of the criteria for meeting Microsoft's BackOffice certification requirements is that an application must be integrated with the Windows NT Server logon. However, this doesn't preclude an application from implementing additional security mechanisms.

1. Abbreviated as I18N because there are 18 letters between "I" and "n."

Security for the server application begins with the Windows NT logon service. To implement the Windows NT logon service, the application must execute as a local service under the Windows NT Server system. When the server boots, the server application will log on using the security identity of the Windows NT Server system Administrator user account.

For the client portion of the application, use application-level security. With NonStop TUXEDO applications, you have several choices of security levels. These are set in the TUXEDO configuration file — you don't program the security into the server application as such — as described briefly in the next section.

NonStop TUXEDO Security

Setting security in the TUXEDO configuration file to "NONE" (the default) will cause the application to rely only on the native security mechanism of the operating system: specifically, the Windows NT Server logon. In this case, read/write and execute permissions will be enabled according to the settings for the logon user ID. Other settings that can be configured will enable applications to be password protected, or will provide more granular, ACL (access control list) checks per service or per server, for example.

Although security is managed in the TUXEDO configuration file and not in the application logic, the application will still need to include code to handle whatever security mechanism is implemented. For example, if you enable a password-protected application (using APP_PW in the configuration file), you'll need to include code in the application to prompt the end-user for a password and to submit the password to the server process. See the *NonStop TUXEDO System Application Programmer's Guide* for detailed information.

NonStop SQL/MX Security

As with other Windows NT Server system database management systems, NonStop SQL/MX installs as a local service, which means it starts when Windows NT Server is booted, regardless of whether any users have logged on or not. The access privileges for the SQL/MX database map to the Windows NT Server Administrator account, which means that it can access all system resources as needed. Users accessing the database must have access rights to the directories in which the database tables are stored.

In addition to the baseline security features provided by the Windows NT Server operating system, NonStop SQL/MX also supports the GRANT and REVOKE data control language statements. Privileges can be GRANTED to or REVOKED from a database object by its owner. To perform an action — INSERT, DELETE, UPDATE, for example — on a SQL/MX table or view, the owner of the table or view must GRANT privileges to classes of users. See the *NonStop SQL/MX Reference Manual* for details.

Select the Appropriate Platform For Deployment

A final note about solutions design: it's important to choose the right platform — or platforms — for the business solution. In some cases, it may be appropriate to develop a completely new solution to run solely on Windows NT Server clusters, while in other

cases it may be more appropriate to deploy some of an application's functionality on Windows NT Server clusters — for example, in branch offices or other satellite type locations — and deploy other processing on a Tandem NonStop Himalaya server.

Developers must evaluate the business problem, assess the alternative solutions, and determine the appropriate development model for their solution. Tandem anticipates that, given the platform independent design of the NonStop Software product line, developers will have numerous development scenarios from which to choose, including:

- Developing new applications for Windows NT Server clusters
- Developing new applications for the Tandem NonStop Himalaya server platform
- Developing applications (or portions thereof) that may span both platforms and therefore must interoperate to varying degrees
- Migrating applications or portions of applications from one platform to the other
- Developing new applications that will run on Windows NT Server clusters but must remain relatively portable to Tandem NonStop Himalaya servers

In Section 3, Getting Started, you'll find information about developing new applications targeted solely for Windows NT Server clusters using the NonStop Software products.

In Section 4, Designing Portable Solutions, you'll find discussion of some of the issues relevant to developing new applications for Windows NT Server clusters that must remain portable to the Tandem NonStop Kernel. For additional information about developing portable applications, see the Tandem white papers entitled "Achieving the Best of Both Worlds: A Guide to Flexible Application Deployment on Both Windows NT Server and NonStop Himalaya Systems" and "Future-Proofing Enterprise Applications."

Note: White papers and guides are available on Tandem's Web site (www.tandem.com); from the NonStop Software Developers' Page (http://www.nonstopsw.com); or from your Tandem representative. From the Tandem Web site, click on the Search button and enter a few key words from the title to find the white paper you wish to see.

For additional information about interoperability between Windows NT Server solutions and Tandem NonStop Himalaya, see "Windows NT Integration with NonStop Himalaya Servers" or the "NonStop Software Interoperability Guide for Windows NT and NonStop Server Systems," available from the Tandem NonStop Software Developers' Page. From the NonStop Software Web site (http://www.nonstopsw.com), select "Developers' Page."

3 Getting Started

The NonStop Software for Windows NT Server product line, comprised of NonStop SQL/MX and NonStop TUXEDO for Windows NT Server, is part of Tandem's strategy to deliver the "best of both worlds" — the Tandem fundamentals combined with Windows NT Server clusters — to its existing and future customers, including Microsoft Windows NT Server application developers, Tandem Alliance Partners, Independent Software Vendors (ISVs), and in-house developers. Products will be released in several phases, the earliest of which is designed for decision support systems. Nonetheless, developers can begin design and prototyping solutions today, for deployment when the fully functional versions of the software are shipped.

This section provides high-level information about NonStop Software for Windows NT Server (hereafter, referred to simply as "NonStop Software") and provides an overview of the application development process in the Windows NT Server environment.

What You Will Need

To get started developing a DSS or OLTP solution targeted for NonStop Software, you'll need little more than an ANSI-compliant C compiler, such as Microsoft Visual C++ (or Borland C++) and a Windows NT Workstation (or Server) platform on which to edit, compile, and debug your prototype application. This subsection provides a summary list of hardware and software requirements, as well as information about some of the tools that may facilitate development.

Hardware and Software Requirements

The hardware requirements your system must meet will depend on which components of the NonStop Software product line you will be using, and whether you want to develop, debug, test, and deploy on the same platform. You can develop and test your application on a single node in the short term, and you can deploy the application on Windows NT Server clusters.

Tandem recommends that your development platform comprise at a minimum a Pentium processor-based Windows NT Workstation with 32-megabytes of RAM and a 2-gigabyte hard-disk drive.

Microsoft Certification

Microsoft certifies specific hardware platforms (manufacturer, model, and configuration) to support the Windows NT Server and Windows NT Workstation operating systems, so be sure you set up your development environment on Microsoft certified hardware.

For single server implementations or development platforms, choose a server from Microsoft's Hardware Compatibility List. Microsoft also recommends servers with the BackOffice logo. See Microsoft's Web site (http://www.microsoft.com/isapi/hwtest/backoffice/hard.idc) for a current list of servers that have the BackOffice logo.

In addition, make sure that all the appropriate device drivers, ROM levels, Service Packs, up-to-date HAL (hardware abstraction layer), and so forth, are installed. For information about Service Packs and other Windows NT Server or Workstation related issues, see Microsoft's Web site for Windows NT Server (http:\\www.microsoft\com\ntserver\info). Also refer to the installation guide that came with your Windows NT Server hardware platform, and any readme files for driver and other hardware-related information.

Clustering Requirements

You don't need a Windows NT Server system cluster to develop or prototype NonStop Software based solutions, nor do you need Microsoft's MSCS. However, developers who may be interested in exploring or experimenting with the MSCS API should contact Microsoft or Tandem (Tandem is a Microsoft Windows NT Server and Windows NT Cluster Server reseller) for information about Microsoft Cluster Server.

For clusters, Microsoft endorses only servers from the Microsoft Cluster Server Hardware Compatibility List. Be sure to consult Microsoft for this list. Hardware requirements to support MSCS are very specific, including types of hard-disk drives supported, the type of disk-drive controller supported, and how the Windows NT Server system should be configured. Your development platform will need to meet these requirements for MSCS, in addition to the requirements for basic Microsoft Windows NT Server platform certification. See Appendix A, Windows NT® Server Cluster Notes, for more details about Windows NT Server and MSCS.

Note: Tandem's S-series Windows NT servers (Tandem S-1000 and Tandem S-1000RM) are certified for both Windows NT Server 3.51 and Windows NT Server 4.x, as well as for the MSCS beta program.

Tandem NonStop Software Certification

Tandem NonStop SQL/MX and NonStop TUXEDO both require Pentium Pro-based Intel SMP hardware. Because Tandem NonStop Software is designed for enterprise-class applications, it is recommended that developers choose hardware platforms certified by Microsoft to run Microsoft Windows NT Server 4/E. Microsoft's minimum requirements include:

- Pentium 90 MHz or higher processor for Intel or compatible systems
- 64 MB of RAM
- 500 MB of available hard-disk drive space (4-GB hard-disk drive recommended)
- VGA, Super VGA, or video graphics adapter compatible with Windows NT Server 4.0
- CD-ROM drive
- Microsoft Mouse or compatible pointing device

However, the base hardware enclosure does not require Microsoft Windows NT Server/E, nor does it require the Microsoft Cluster Server software. (Neither is shipping at this time, but both are expected to be shipping soon.) NonStop Software will run on Windows NT Server 4.x and above. Check the NonStop Software Developers' Page for

current Tandem configuration and certification requirements. From Tandem's NonStop Software Web page (http://www.nonstopsw.com), select "Developers Page."

Setting Up the Development Platform

Install the Windows NT Server (or Workstation) operating system on hardware certified by Microsoft, choosing TCP/IP as the network protocol. Refer to Microsoft's Windows NT Server or Windows NT Workstation installation guide for current requirements, supported platforms, and other configuration details. Here is a short overview:

- If you're installing your server from the ground up, be sure to have on hand all video display and other device drivers as needed to ensure a working installation.
- Apply the appropriate Service Packs. See Microsoft's Web site for details. As of June 1997, the current Service Pack is Service Pack 3 (nt4ssp3i.exe for the i386 platform).
- If you have a printer connected to the server, install the print queue later, after the server is up and running, because it will then be easier to debug any installation problems you might have.
- Once the Windows NT Server is properly configured and working, start up and shut down the server once or twice to ensure that you have no configuration problems.

Install the NonStop Software Components

Once the Windows NT Server (or Workstation) base platform is operational and has the appropriate level of software patches (Service Packs) applied, you can install the NonStop Software for Windows NT Server components. Here's a summary of the process:

- Install the application development environment (C/C++ or COBOL) according to the software vendor's instructions.
- For database applications, install and configure NonStop SQL/MX. NonStop SQL/MX uses the standard Windows-based InstallShield utility, which provides a series of Wizards that guide you through the installation. NonStop SQL/MX is installed from a single node, but it will install on all nodes of a cluster.
    - All components, including client-side components (DLLs, ODBC driver, C preprocessor) are installed on the server.
    - See the *NonStop SQL/MX Installation Guide*, available on the NonStop SQL/MX CD ROM, for detailed installation instructions.
- For transaction processing applications, install and configure the NonStop SQL/MX database and NonStop TUXEDO.
    - For prototyping and development, install the developer's version of NonStop TUXEDO. The runtime version can be used for testing or deploying on a single Windows NT Server system.
    - If you wish to use a database other than NonStop SQL/MX for your prototype, see Near-Term Implementation Issues on page 3-10 for more information.

Client/Server Development Issues

In this section you'll find information about application development environments, client development tools, and overviews of interprocess communications mechanisms supported on the Windows NT Server platform. If you wish to develop applications for both Windows NT Server systems and Tandem NonStop Himalaya servers, see Section 4, Designing Portable Solutions, in addition to this section.

Application Development Environments

For optimal cross-platform development opportunities, Tandem recommends coding your application using the ISO/ANSI C or ISO/ANSI COBOL85 programming language. The compiler you choose will depend on a variety of factors, including your development platform, your target platform, whether these two are the same or different, and whether you want to develop a cross-platform (Windows NT Server and Tandem Himalaya server) solution.

Client Development

The most popular application development environments for the client applications by far are Microsoft Visual BASIC and Sybase/Powersoft PowerBuilder. For NonStop SQL/MX, you can use C++ with ODBC and C with either ODBC or embedded SQL.

Optional Client/Server Development Tools

There are also numerous third-party development tools for developing client/server applications for TUXEDO application environments, such as:

- BEA Builder
- BEA Jolt
- BMC Patrol
- Borland Delphi
- Dynasty DYNASTY
- Microsoft Visual Basic
- Sybase/Powersoft PowerBuilder
- Unify ACCELL/TP

This list comprises a range of tools, from application development environments, such as Microsoft Visual Basic, to Dynasty, which generates the code required to interface to the BEA TUXEDO API. Refer to specific vendors for more information about product features and functionality.

Code Management and Version Control

For code management in an enterprise application development environment, there are several products on the market, some of which are listed below, that are supported on

the Windows NT Server and other platforms. These code management tools provide a wide range of functions, from basic source control and version management, to more sophisticated management of multiple code bases and problem tracking. Investigate and pick the tool that best meets your needs.

- Pure Atria ClearCase
- Source Integrity MKS
- Microsoft's Visual SourceSafe
- Intersolv PVCS
- Data Design Systems RMS (Revision Management System)
- Network Concepts Control CS

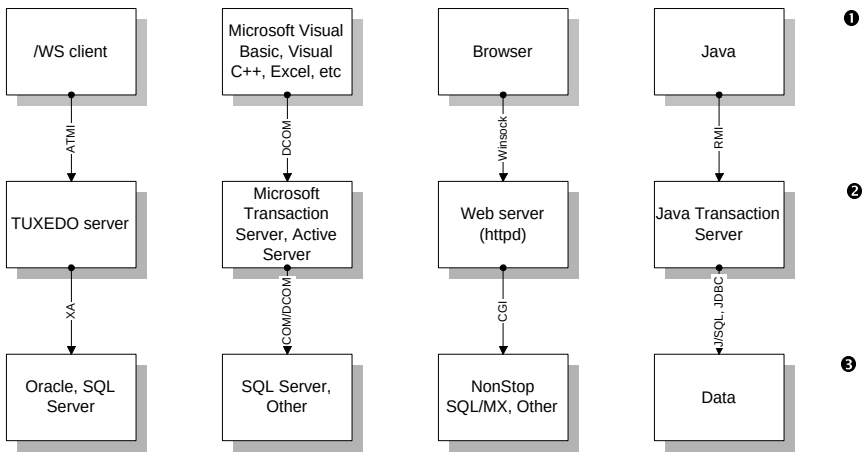
Interprocess Communications

Microsoft Windows NT Server supports several client/server IPC mechanisms for distributed communications. This is a general discussion of three recommended transport-independent mechanisms:

- DCOM
- RPC
- Winsock

All three mechanisms can be implemented over TCP/IP, and in some cases, over other protocols. However, as is the case with Windows NT Server/E and MSCS, only TCP/IP is supported by NonStop Software.

Figure 3-1. Client/Server Example Implementations



Windows NT Server systems support numerous interprocess communication (IPC) mechanisms. The examples above are not all architecturally equivalent; some are lower level than others. For example, ATMI

makes use of lower-level sockets programming functions.

These mechanisms are not architecturally equivalent; that is, DCOM itself is built on top of Microsoft's RPC implementation, and RPC can use Winsock (and other protocols) for transport. Furthermore, DCOM is a complete architecture and implementation which encompasses the network transport as well as an underlying architectural model built on distributed components (akin to object orientation, but not precisely), which may make it less open in certain circumstances. Windows NT 4.0 and Windows 95 support DCOM, and both client and server in a DCOM application must be configured for DCOM support (in the Registry).

In addition to these distributed client/server communication mechanisms, you can use higher-level APIs, such as NonStop TUXEDO ATMI API with /WS (for workstation support).

For more detailed information about any of the IPC mechanisms provided by Windows NT Server, see the Windows NT Server Resource Kit, specifically, the *Windows NT Server Networking Guide*. In addition, the Microsoft Developer Network Library contains extensive and detailed technical tutorials about developing client/server applications using a variety of mechanisms and frameworks.

DCOM

Microsoft's DCOM (distributed component object model) is the result of several years of evolution of the OLE programming model, and is a network-extended version of Microsoft's COM (component object model). COM is a binary standard that specifies language-neutral calling conventions among components[1]. Components written in one language can call components written in another language. In DCOM, components can reside at different locations on the network.

DCOM (formerly known as network OLE) is supported in Windows NT Server and Workstation 4.0[2], and has also been released as beta code for Windows 95. DCOM must be configured at both client and server. The Windows NT (and Windows 95) Registry contains the location information that the COM objects need to find each other over the network; DCOM consults the Registry to find out where the called component is located.

DCOM underlies much of Microsoft's operating system and services infrastructure, and is the application development model that Microsoft will be using for its future product development efforts. As an example, Microsoft Transaction Server (MTx, formerly code-named "Viper"), is built completely on a foundation of COM/DCOM. Developers can create solutions comprised entirely of components using the DCOM model to be run in Microsoft Transaction Server.

Tandem recommends using DCOM for client/server development unless you plan to migrate portions of the application to other platforms, such as Tandem NonStop Himalaya servers.

For more information about DCOM, see Microsoft's "DCOM Technical Overview" White Paper, available at http://www.microsoft.com/msdn/.

1. A component is an executable program image which can have multiple interfaces. Each interface has its own collection of methods (in the object-oriented sense of the word). An object is an instance of an component.

2. With Service Pack 2.

RPC

The Microsoft Windows NT Server RPC implementation is compatible with OSF/DCE (distributed computing environment), although not completely identical. Unlike DCOM, RPC (remote procedure call) as the name implies is based on a procedural paradigm, not an object paradigm. RPC extends the local procedure call paradigm occurring within a single process to processes in different network locations. A process in one address space calls a process in another address space; a value is returned, just as if the two were running in the same address space. The called procedure either sends a return message or an exception so that the caller can respond accordingly. In RPC, processes can be running on different machines or on the same machine without affecting the coding model.

Clients and servers both make calls to stub processes, which contain the location information they need to locate the called process. An interface definition language (IDL) compiler, in this case, Microsoft's IDL (MIDL), is used to generate the client and server stubs and headers from the interface definition and application configuration files that the developer creates.

In addition to the interface definition files, the developer must also write code for the client to manage its connection to the server, and write code for the server processes. These files are then compiled and linked to the RPC runtime library to produce the distributed application.

Microsoft RPC is included with the Win32 SDK and is a C/C++ programming interface. The 32-bit MIDL (Microsoft Interface Definition Language) compiler is installed during Win32 SDK setup. This also installs the RPC runtime libraries. Unfortunately, RPC cannot be accessed directly by higher level tools, such as Visual Basic; it needs an object oriented wrapper in order to be implemented by such tools.

For more information about RPC programming, see the Microsoft Developer Network Online (http://www.microsoft.com/msdn).

Winsock

The Microsoft Windows Sockets API (Winsock) was initially developed for Windows and UNIX applications interoperability. The initial version provided an abstraction layer for programming to the TCP/IP protocol family, but the design is such that it is essentially protocol independent. Basically, Windows Sockets (Winsock 2) can be used to develop a network-aware application supported over TCP/IP.

For more information about Winsock and Windows Sockets programming, see Microsoft's technical article "Developing Transport-Independent Applications Using the Windows Sockets Interface." The WinSock 2 SDK for Windows 95 is available from Microsoft at http://www.microsoft.com/msdn. For WinSock 2 API information, including the API, you can download the specification from ftp://ftp.microsoft.com/bussys/winsock/winsock2/ or ftp://ftp.intel.com/pub/winsock2/.

Internet Technologies and How They Fit

Microsoft Windows NT Server 4.0 includes an integrated Web service, Internet Information Server (IIS), that supports several different Web-to-application-server

application programming interfaces (APIs). These interfaces, which support calls between the Web server or the application service and the database or the server that's hosting the database, are discussed briefly in the following subsection along with related client Internet technologies.

Server APIs

The common gateway interface (CGI) is Web-server API. CGI can send program parameters using environment variable, and keeps track of context using cookies and URL encoding. When the client clicks a URL that corresponds to a CGI script, the CGI script can:

- start a transaction
- call a local transaction manager
- ask the other servers participating in the transaction to join

ISAPI (Internet Server API), a Microsoft API that communicates securely between services, and NSAPI (Netscape Server API) are two other APIs supported by both Microsoft IIS and other Web servers.

Java

Java is a general-purpose, object-oriented programming language developed by Sun Microsystems. Key design goals for Java were that it be portable and lightweight; thus, it has become an ideal programming environment for the Internet. Java is designed to be architecture neutral and completely portable. It runs on any platform for which there's a Java Virtual Machine (JVM, a runtime interpretive environment). *Applets* are Java applications that require an instance of a JVM in which to execute. Netscape Navigator, Internet Explorer, other browsers, and other client programs in general can incorporate the JVM mechanisms for purposes of running Java applets when they need to.

Java *servlet*s provide an alternative to a C application and the CGI interface between a service and a Web server. Peer-to-peer services require inter-applet communication across a network. Remote method invokation (RMI) is an API that enables peer-to-peer methods of remote Java objects to be invoked from other Java Virtual Machines. Conceptually similar to RPC programming, the RMI code generator (rmic) creates a stub and a skeleton.

The JVM doesn't access system-level services, so there's a level of built-in security. For example, if a virus were downloaded, it couldn't read or write to the hard-disk drive, since these are system-level services. (This is one of the differences between Java and ActiveX, which does access system-level services.)

The emergence of Java has initiated numerous other development efforts aimed at bringing Java to all aspects of enterprise-wide computing, including Internet and Intranet access and interaction with corporate data and operational systems. For example, J/SQL, a specification for Java interface for embedded SQL sponsored jointly by Oracle, IBM, and Tandem, is intended to simplify Java access to data by providing automatic mapping between SQL data types and Java objects. J/SQL complements JDBC (Java Database Connectivity), the standard interface for enabling Java applets to access relational database management systems; JDBC is similar to ODBC (Open

Database Connectivity). For more information about Java and database access through JDBC, see “JDBC™ Guide: Getting Started” available on the JavaSoft Web site (http://http://java.sun.com/products/jdk/) along with numerous other publications.

Jolt extends the TUXEDO product capabilities to Java applications and applets. Jolt is a set of software components that enable Java programmers to request TUXEDO application services from Web browsers that support the Java Virtual Machine, without needing to know detailed transaction semantics. For more information, see BEA Systems Web site (http://www.beasys.com).

Java applications (server applications) will be supported on Windows NT Server systems and Tandem NonStop Himalaya servers.

ActiveX

ActiveX is an umbrella term for COM/DCOM-based Microsoft technologies that enable developers to create interactive Web content. Although ActiveX was developed initially by Microsoft, control of the ActiveX standard has been moved to the Active Group[1], an open standards body. Microsoft is working with Metrowerks to support ActiveX on the Macintosh platform, and is working with Bristol and Mainsoft to support ActiveX on UNIX platforms.

Active Client (which is included in Internet Explorer 3.0 and Internet Explorer 4.0) is essentially an Internet component that supports HTML. ActiveX Controls are the interactive programmable objects that can be embedded in Web pages to provide interactive and user-controllable functions, similar in function to Java applets, for example.

Microsoft’s ActiveX Server Framework provides a number of Web server-based functions, including security, database access, and server-side technology extensions, including ISAPI, ActiveX server controls, ActiveX server applications, activeX server filters, and ActiveX server scripts. Active Scripting controls the integrated behavior of several ActiveX controls or Java applets from the browser or server.

For more information about ActiveX and related technologies, see Microsoft’s Web site (http://www.microsoft.com/msdn/activex).

TIP (Transaction Internet Protocol)

Standard APIs enable application and database system portability between TP monitors; standard protocols enable interoperability between TP monitors. Until recently, OSI TP was the only two-phase-commit protocol developed by an independent standards body, but it has not garnered wide industry acceptance. The Transaction Internet Protocol (TIP) is an Internet draft specification for a simple two-phase commit protocol which will enable distributed transaction processing between heterogeneous transaction processing monitors. TIP is an important emerging standard being jointly developed by Tandem and Microsoft.

1. The Active Group is a consortium of software and systems vendors dedicated to promoting and gaining widespread acceptance of ActiveX core technologies. The Active Group is an authoring group working under the auspices of The Open Group (which encompasses X/Open and OSF). The steering committee for the Active Group is composed of 17 vendors, including Adobe Systems, ATT, Borland International, Computer Associates International, Digital, Hewlett-Packard, Microsoft Corporation, NCR Corporation, Powersoft-Sybase, and SAP AG, to name just a few.

TP monitor vendors and others who wish to can implement TIP in their products to support heterogeneous two-phase commits. For example, Tandem is developing a TIP gateway product to enable two-phase commit between Tandem NonStop TUXEDO products, Tandem NonStop TM/MP, and any other TP monitor that implements the Transaction Internet Protocol.

To stay informed about this important emerging protocol standard, subscribe to the TIP mailing list by sending email to listserv@tandem.com with the words “subscribe tip” embedded in the message body. Copies of the current draft protocol specification are available at Microsoft and Tandem Web sites, and from the InterNIC (Internet Network Information Center) — the specific document can be found at ftp://ds.internic.net/internet-drafts/draft-lyon-itp-nodes-02.txt.

For More Information

Microsoft Developer Network (MSDN), now available on the Internet (http://www.microsoft.com/msdn/), has numerous techical articles, utilities, code samples, white papers, technical overviews, and SDKs for Microsoft Windows NT developers. In addition, Microsoft hosts Web sites for each of its product and platform lines. For many of the implementation details about developing and coding client/server applications in the Microsoft DCOM/ActiveX paradigm, Tandem recommends that you refer to the Microsoft Developer Network.

Near-Term Implementation Issues

The NonStop Software for Windows NT Server product line is being released in several phases, some of which may be beta releases or early adopter versions of the NonStop SQL/MX and NonStop TUXEDO for Windows NT Server products. Depending upon the versions of these two products that you may be using for prototyping, designing, or testing purposes, you should be aware of some short-term implementation details. When FCS (first customer ship) versions of NonStop SQL/MX and the NonStop TUXEDO for Windows NT Server 2.0 are released, these implementation issues will be irrelevant, but developers working with versions prior to the FCS should review this section.

Be sure to check Tandem’s NonStop Software Web site (http://www.nonstopsw.com) for up-to-date information.

Using NonStop SQL/MX Only

The NonStop SQL/MX product is an ISO/ANSI-92 SQL compliant database that can be used for both decision support and OLTP applications. Many developers may wish to implement a two-tier client/server model, using ODBC from the client application for accessing NonStop SQL/MX data. Database application developers, such as data warehouse or “data mart” developers using NonStop SQL/MX alone for development, can:

- Use ODBC as the call-level interface to access data

For OLTP applications, developers should:

- Use ISO/ANSI C as the programming language for server applications

- Use ISO/ANSI-92 SQL (embedded) for transaction demarcation and processing

If you stay within these guidelines, you should have little problem implementing your DSS application when the DSS version of NonStop SQL/MX is released, or implementing your OLTP application when that version is released (see Table 3-4 on page 3-16 for planned releases and features).

However, developers who are familiar with Tandem's NonStop SQL/MP product (for the Tandem Himalaya server) should be aware that the NonStop SQL/MX relational database product is a completely new product, based on the current standards (ISO/ANSI-92 SQL), and is different from NonStop SQL/MP. See the section entitled For NonStop SQL/MP Developers on page 5-14 for more information.

In addition, developers who wish to implement NonStop SQL/MX with the NonStop TUXEDO product — in order to gain scalability and manageability of a distributed application, for example — should read the following section for some implementation strategies.

Using NonStop TUXEDO and NonStop SQL/MX

If you're developing an application that must scale to support hundreds or thousands of users, or one in which data or application components are distributed across numerous servers, services, or resource managers, you should develop your application using a TP monitor, such as NonStop TUXEDO.

The NonStop TUXEDO TP monitor provides a framework for distributed computing because it provides several APIs for distributed communications, for distributed transactions, and for managing resources. Before discussing the specifics of the NonStop TUXEDO product, the section starts with an overview of distributed transaction processing monitor basics.

TP Monitor Overview

The DTP (distributed transaction processing) model promulgated by X/Open (now part of The Open Group) delineates and defines many of the standard functions provided by TP monitors. The DTP model divides transaction processing systems into components based on function. The three primary functions are:

- managing transactions
- managing resources, including databases
- managing communications among applications, and among clients and servers

Each of these functions is provided by a subsystem named accordingly: that is, transaction manager, resource manager, and communications manager, respectively.

Transaction Manager Interfaces

The transaction manager subsystem provides the application programming interface that is used to delimit transaction content. This interface provides functions to mark the start and end of a transaction, and is also used to either commit, roll back, or abort the

transactions. In the X/Open's DTP model, the interface that supports these functions is the TX interface.

BEA TUXEDO supports the TX interface, and additionally provides the "Application-Transaction Manager Interface," or ATMI. X/Open has incorporated the ATMI interface in its DTP model (as XATMI), in addition to TX.

Resource Manager Interface

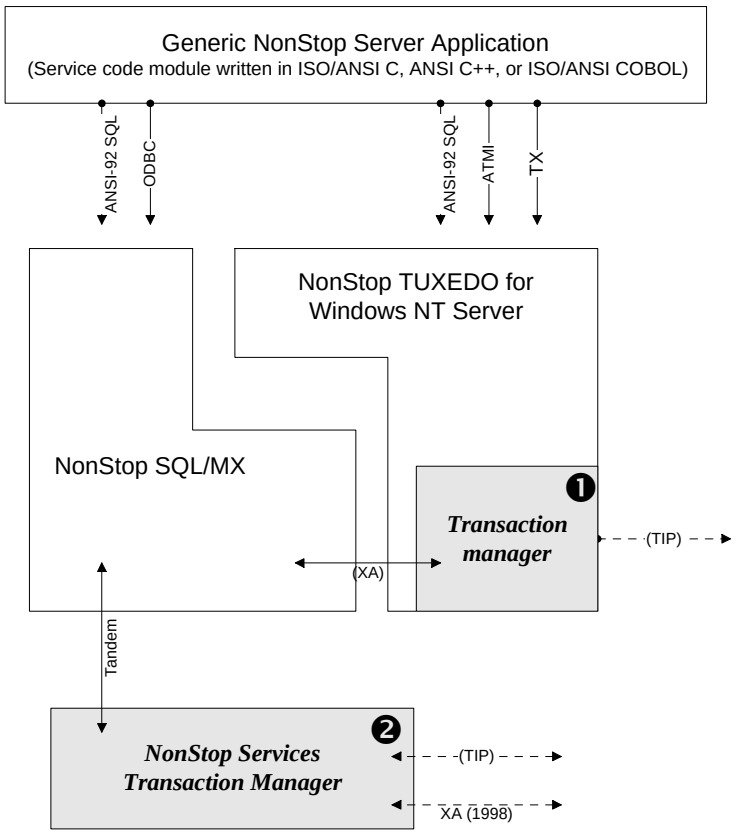
In addition to the interface between the application and the transaction manager, if the transaction manager is going to interface to a resource manager, such as a database management system, there must be an interface between these two subsystems as well. In the DTP model, this is the XA interface. A given database management system can support the XA interface or another interface integrated with a particular transaction management subsystem, or both.

A database management system that supports XA typically includes an XA library that can be compiled as part of the application. Calls from the application are made using the database's native call-level interface, and transparent to the application, the XA library couples the transaction manager and the resource manager — the database — together for negotiating a two-phase-commit protocol. Thus, a TP monitor and a database that both support XA can work together to coordinate a transaction.

Interfaces Supported by Tandem's TP Monitor Software Products

Tandem's NonStop TUXEDO software product for NonStop Himalaya servers is essentially the same as BEA's TUXEDO product, with the notable difference being that Tandem has replaced BEA's transaction manager subsystem with the Tandem NonStop TM/MP software product, a parallel transaction manager designed for clustered systems. NonStop TM/MP brings many benefits, including the distributed transaction logging facility with its virtual single log image (for greater ease of recoverability).

Figure 3-2. Transaction Manager and Resource Manager Interfaces



A TP monitor, such as NonStop TUXEDO, typically encompasses features of transaction management, resource management, and communications management. In this figure, only the transaction managers and resource managers are shown.

Why is this important? Because the NonStop TUXEDO on Windows NT Server product will implement this same facility as an infrastructure component, known as NonStop Services Transaction Manager. This subsystem will not be XA compliant in the short-term, but in the long-term, will be XA compliant, thus enabling developers to use NonStop TUXEDO with other vendor databases in addition to NonStop SQL/MX.

These two distinct transaction managers are shown in Figure 3-2:

1. The native BEA TUXEDO transaction manager, which is XA compliant.

   The interface implemented in Tandem NonStop TUXEDO Release 1.0 is identical to BEA TUXEDO transaction manager, and is therefore also XA compliant.

2. The NonStop Services Transaction Manager is not currently XA compliant.

   NonStop SQL/MX uses NonStop Services Transaction Manager in all releases, but through an integrated interface.

NonStop TUXEDO will implement NonStop Services Transaction Manager, without XA compliance, in NonStop TUXEDO Release 2.0.

The NonStop Services Transaction Manager will support the XA interface in mid-1998.

Regardless of a particular vendor's implementation of the BEA TUXEDO APIs, any TUXEDO implementation can be integrated with an XA or non-XA compliant database (or other resource manager).

How XA Compliance Affects Application Development

The application you write doesn't make direct use of the XA interface — there are no "XA APIs" to which developers can code. Only the transaction manager and resource manager use this interface to coordinate a two-phase-commit transaction.

However, the fact that NonStop SQL/MX in its initial release is not XA compliant, and the fact that NonStop TUXEDO for Windows NT Server 1.0 is XA compliant, will necessitate configuration workarounds in order for these two products to work together, just as BEA TUXEDO requires configuration modifications to work with non-XA compliant databases.

To use NonStop TUXEDO for Windows NT Server (Release 1.0) with NonStop SQL/MX, you must do one of the following:

- Use unaudited tables so that the database log file doesn't attempt to write to the transaction manager's log file. The default is for NonStop SQL/MX tables to be audited, so you'll need to create UNAUDITED tables in the short-term if you want to use NonStop SQL/MX with NonStop TUXEDO for Windows NT Server 1.0.
- Precede the transaction statements with comment codes so that they effectively are non-operational.
- Use SQL transactions rather than ATMI (or TX) transactions. By doing so, the transaction manager subsystem (which provides the TX and ATMI interfaces) won't be invoked.

In addition, there are basic setup issues including modifications to the TUXEDO configuration file that you must take into account for interfacing with any non-XA-compliant database. For additional details, see the *TUXEDO Application Development Guide.*

When the FCS of NonStop TUXEDO for Windows NT Server 2.0 and NonStop SQL/MX are both released, you can then reverse the process for these solutions: for example, remove the comment delimiter from your code to test the transactional semantics, or replace the SQL transaction demarcation verbs with ATMI transaction demarcation.

These are interim measures that are only required using specific versions or releases of the two NonStop Software for Windows NT products, as discussed earlier. As shown in Table 3-1, the NonStop TUXEDO for Windows NT Server (Release 2.0) will use the integrated interface (from the Tandem NonStop Transaction Manager subsystem) in addition to the native BEA TUXEDO transaction manager, so XA compliance won't be an issue, regardless of the database involved.

Table 3-1. Transaction Managers and XA Compliance

	XA
BEA TUXEDO	yes
IBM CICS/Transaction Server	yes
NonStop TUXEDO	no
NonStop TUXEDO for Windows NT Server (Release 1.0)	yes
NonStop TUXEDO for Windows NT Server (Release 2.0)	no
NonStop TM/MP	no
Microsoft Transaction Server	no
Transarc/IBM Encina	yes

An alternative is to develop and test your application using a resource manager that supports the XA interface. Table 3-2 shows a short list of some of the current databases on the market and XA support. Note that this is by no means a complete list of products, but is offered for illustration purposes.

Table 3-2. Resource Managers and XA Compliance

	XA
IBM DB2 for Windows NT	yes
Informix	no
NonStop SQL/MX 1.0	no
NonStop SQL/MX 2.0	yes
NonStop SQL/MP	no
Oracle	yes
SQL Server	yes

For more detailed and current information about developing and testing applications for NonStop Software, see Tandem's NonStop Software Developers' Page. From the NonStop Software home page (http://www.nonstopsw.com), select "Developers' Page."

NonStop Software for Windows NT Server Features

The NonStop Software product set marks Tandem's entry in the Windows NT Server cluster software marketplace. Tandem is using a phased approach to product release to enable developers to get started early in their development efforts. That said, it's important to recognize that the initial product releases may not include complete product functionality. Table 3-3 and Table 3-4 show features sets and capabilities provided and planned for the NonStop Software products in the upcoming release phases. The features

lists were compiled during June 1997, and are subject to change. Be sure to check the NonStop Software Web site for current information.

Table 3-3. NonStop TUXEDO for Windows NT Server Features

	Release 1.0	Release 2.0 (Beta)	Release 2.0 (FCS)
Based on BEA TUXEDO version	6.3	6.3	6.3
Supports Microsoft Visual C++ 4.x	yes	yes	yes
Supports Borland C	yes	yes	yes
Supports COBOL	yes	yes	yes
Support for ServerNet® interconnect mechanism	no	yes	yes
ServerNet® required	no	no	yes
NonStop Software clustering	no	yes	yes
Microsoft Cluster Server 2-node fail over	na	na	na
Jolt	yes	yes	no
XA interface	yes	no	yes
NonStop SQL/MX integrated interface	no	no	yes

Table 3-4. NonStop SQL/MX for Windows NT Server Features

	Beta	DSS Pre-release	OLTP FCS
Precompiler for C	yes	no	no
Precompiler for COBOL	no	yes	yes
Precompiler for C++	no	no	yes
Support for ServerNet® interconnect mechanism	no	yes	yes
NonStop Software clustering	yes	yes	yes
Microsoft Cluster Server 2-node fail over	na	na	na
Microsoft Cluster Server n-node clustering	na	na	na
ODBC	yes	yes	yes
DataBlade technology	no	no	yes
JDBC	no	no	no
J/SQL	no	no	no
XA interface	no	no	no

4 Designing Portable Solutions

One of the key benefits of NonStop Software products is that you can develop your application in a uniprocessor, nonclustered Windows NT environment — for example, on an Intel based computer running Windows NT 4.0 Workstation — and then compile, link, and deploy the application on either Windows NT Server clusters, Tandem NonStop Himalaya servers, or both.

Nonetheless, given that each systems' respective processor architecture and operating system are inherently different, there will be some issues that developers must concern themselves with to ensure that the applications they develop will port as seamlessly as possible from Windows NT Server clusters to Tandem NonStop Himalaya servers. This section identifies some of these issues and presents some high level design guidelines for developers who wish to target both Windows NT Server clusters and NonStop Himalaya servers.

What Is a "Portable" Application?

A portable application is an application running on one platform that can be recompiled for another target platform without a significant amount of recoding effort. The definition of the word "significant" is always relative, and the key question when deciding whether to port an application or not is whether the expected benefits will outweigh the costs (time, labor, implementation issues, potential failures, and so forth). Will the expected outcome be worth the effort? Or can a portion of the application be ported more cost-effectively than the entire application?

Determining the benefits, costs, and potential future impacts of porting an application, good and bad, and deciding whether to port or not, in part or in whole, requires a case-by-case evaluation. Nonetheless, "porting" an application is an activity that is best done when the application includes a large percentage of "portable" code. Portability should be a stated design requirement from the beginning, not after the fact.

One of the basic steps developers can take to design portable applications is to choose open, industry standard application programming interfaces and protocols that are available on multiple platforms. The "industry standards" can be either de facto or de jure standards, as long as more than one vendor has implemented the API or protocol in question.

But choosing industry standard APIs and protocols is just the beginning. For an overview of the engineering approaches one can take to develop portable applications, see the Tandem White Paper entitled "Future-Proofing Enterprise Applications," available on Tandem's Web site (http://www.tandem.com).

Portability Compared to Interoperability

The concept of "portability" is sometimes confused with "interoperability." Interoperability is a characteristic that enables two entities in a distributed computing environment, such as an application or parts of an application, to interact and share information or share processing. This section focuses on developing applications for Windows NT Server clusters based on the assumption that these applications should be

portable to Tandem NonStop Himalaya servers; interoperability between the two platforms is not part of the discussion. For high-level information about interoperability between Tandem NonStop Himalaya servers and Windows NT Server systems, see Tandem ServerWare business unit's publication entitled "NonStop Software Interoperability Guide for Windows NT and NonStop Server Systems," available on Tandem's NonStop Software Developers' Page. From Tandem's NonStop Software home page (http://www.nonstopsw.com), select "Developers Page."

An Approach to Multiplatform Solutions

One key to designing solutions for multiple platforms is adhering to industry standards. Adherance to leading industry standards has been an important design criterion of the NonStop Software product line since its inception, which is why there are no "NonStop Software APIs." As mentioned earlier and as highlighted in Figure 2-2, Server Application (Middle-Tier) Model, on page 2-8, the APIs that you'll use to develop your business solution are the same industry-standard, open APIs that you might have used with database products from other vendors, or with BEA TUXEDO on any number of other platforms. These include:

- ANSI C++
- ATMI[1]
- ISO/ANSI C
- ISO/ANSI COBOL
- ISO/ANSI-92 SQL
- ODBC
- TX

If you stay within the confines of these APIs and avoid using proprietary extensions, your code should be relatively portable between platforms. The following subsections include some additional high-level guidelines.

Application Design Issues

Regardless of the specific implementations details (programming language, compiler, and so forth), follow these high-level guidelines for any application that you wish to target for both Windows NT Server clusters and Tandem NonStop Himalaya servers:

- Design your application around open middleware. Use NonStop TUXEDO as a framework for highly scalable applications and to ensure that the code base will easily port among platforms (over 30 different platforms) that support the TUXEDO API.
- For whichever language you choose, use a compiler that complies with the ISO/ANSI standards. For example, use an ANSI-compliant C or COBOL compiler.

1. Although ATMI itself is not the result of an industry standards organization, the ATMI API and other APIs originated by the TUXEDO sofware product are the basis for the X/Open's DTP model standards, including XATMI and TX.

- Use ODBC or embedded SQL (and not C or COBOL I/O verbs) for data access.
- Do not use shared memory because it prevents an application from scaling on a clustered platform. Use message-based interprocess communication mechanisms instead.
- Avoid hardware dependencies, such as data layout or alignment. Avoid features that are not available on both platforms, or isolate platform-specific code into small modules that can be completely replaced for the target platform. Alternatively, if the amount of platform-specific code is minimal, you can use conditional compile — for example, #ifdef in C and C++ — but use them sparingly.
- Do not use the Microsoft Foundation Class (MFC) libraries (included in Microsoft Visual C++ and supported by many other C/C++ development environments) for the server application because the NonStop Kernel doesn't support MFC libraries. (However, you can use MFC libraries for your client application; MFC libraries provide many of the graphical user interface features you'll need for Windows 95 or Windows NT Workstation clients.) Rather, use a product designed for cross-platform and muliple platforms, such as Rogue Wave Tools.h++ for C++.
- Engineer for portability by means of abstraction and isolation. For example, create an abstraction library for platform-specific services which can be used by other processes and programs.
- If you plan to develop first for the Windows NT Server platform, keep in mind that the NonStop Kernel platform doesn't support the Microsoft Windows dynamic load library (DLL) paradigm. If you want to use DLLs on your Windows NT Server implementation, isolate these functions so that they will be easy to recompile and link into a single object file for the NonStop Kernel. (These functions probably already are isolated, given the fact that DLLs comprise separate code modules.)
- As you compile and debug your code, eliminate compiler warnings as much as possible. Although a compiler warning may not cause a problem on the initial target platform, it may lead to additional problems when re-compiled for the second target platform. It's good practice to eliminate any warnings so that they don't lead to other problems.
- Use memory testing tools on Windows NT Server systems (for example, BoundsChecker) to detect memory leaks.
- Avoid X/Open UNIX system features that are not available on the PC platform (for example, sys\time.h header).

Development Process For Multiple Targets

There are a couple of different approaches that developers can take for multiplatform development. One approach is to select the platform that will be the initial target and develop to this platform first, using the development environment that is best suited for the primary target, and then recompiling for the secondary platform later.

However, rather than use two different development environments aimed at different target platforms, whenever possible it's preferable to select a single development environment that can generate code for both platforms. The benefit of this approach is

that there's less of a learning curve for developers, who will use the same editor, source control, and same basic development paradigm for both target platforms.

For C/C++ developers, Tandem offers an integrated development environment, the Tandem Development Suite (TDS) Using Borland Software product, which will support development for both NonStop Kernel and Windows NT Server targets. TDS is a 3GL development environment that enables C/C++ programmers to develop applications for multiple Tandem platforms. It provides integrated editing, compiling, linking, and project manager/automated build tools for use in a Windows NT or Windows 95 environment.

The TDS integrates Borland C++, Tandem native C/C++ cross-compilers, Tandem Extensions to Codewright, Visual Inspect, the Native Mode C Migration Tool, and ftp into a consistent and easy to use tool set that increases productivity and provides the performance gains associated with PC based compilation.

More detailed recommendations for tools for editing, compiling, testing, and managing source code for both NonStop Kernel and Windows NT Server are contained in Table 4-1 (for C/C++ development) and in Table 4-2 on page 4-6 (for COBOL development).

ISO/ANSI C and ANSI C++

In addition to the recommendations listed in Application Design Issues on page 4-2, be sure to observe these guidelines when developing C/C++ applications for either NonStop Kernel or Windows NT Server systems:

- If NonStop Kernel is the primary platform, use native C/C++ (D4x/Gxx) for code base because it has best standards support, best performance, most commonality between platforms. In addition, it is implemented in a PC-based development environment.
- If Windows NT Server is the primary platform, use Microsoft Visual C++
- Use Rogue Wave Tools.h++ class libraries for C++ programs for either platform.
- In Microsoft Visual C++, use the /Za compiler option to enable strict ANSI compatibility checking.
- Use ANSI-model C I/O (file pointers, not descriptors).

If you're recompiling NonStop Kernel code for the Windows NT Server system target, be sure to:

- Undefine _TANDEM_SOURCE and OSS_SOURCE macros
- Remove the EXTENSIONS pragma
- Isolate platform-specific features using conditional compilation (OS calls, GUI calls, language extensions)
- Avoid case-sensitive names
- Specify text or binary for file opens with fopen()

- Use a standard set of #defines for conditional compilation for:
  - OS calls
  - Big-endian vs. little endian
  - Compilers
  - Language extensions

Table 4-1 provides a summary list of recommended application development environment tools for various source and target platforms.

Table 4-1. C/C++ Application Development Tools

	Development Platform	Target Platform	
		NonStop Kernel	**Windows NT Server**
Edit	PC	Tandem Extensions to Codewright	Tandem Extensions to Codewright, Microsoft Visual C++
	Guardian	TEDIT	na
	OSS	vi	na
Compile/Link	PC	Tandem Development Suite Using Borland Software with Native C and C++ cross-compilers	Microsoft Visual C++ compiler, Tandem Development Suite Using Borland Software, Borland C++ Compiler
	Guardian	NMC, NMCPLUS, nld	na
	OSS	c89, nld	na
Debug	PC	Visual Inspect	Microsoft Visual C++ debugger
	Guardian	Inspect	na
	OSS	Inspect	na
Manage sourcecode	PC	PC-based source management software	PC-based source management software
	Guardian	NonStop Kernel source management software	na
	OSS	NonStop Kernel source management software	na

COBOL

Recommendations for tools for editing, compiling, debugging, and managing source code are listed in Table 4-2. In addition to the general recommendations listed in Application Design Issues on page 4-2, be sure to observe these guidelines:

- Use Micro Focus COBOL for Windows NT Server
- Use native Tandem COBOL85 (D4x/Gxx) for base
- Code to ISO/ANSI COBOL85 standard
- Use standard language features
- Specify SUBSET HIGH, DEB1, SEG2, OBSOLETE directives to identify features that are not portable
- Isolate platform-specific features using conditional compilation (OS calls, language extensions)

Table 4-2 provides a summary list of recommended COBOL application development environment tools for various source and target platforms.

Table 4-2. COBOL Application Development Tools

		Target Platform	
	Development Platform	**NonStop Kernel**	**Windows NT Server**
Edit	PC	Tandem Extensions to Codewright	Tandem Extensions to Codewright; Borland IDE editor; Micro Focus COBOL editor
	Guardian	TEDIT	na
	OSS	vi	na
Compile/Link	PC	~	Micro Focus COBOL compiler and linker
	Guardian	COBOL, NMCOBOL, nld	na
	OSS	c89, nld	na
Debug	PC	Visual Inspect	Micro Focus COBOL debugger
	Guardian	Inspect	na
	OSS	Inspect	na

Table 4-2. COBOL Application Development Tools

		Target Platform	
	Development Platform	**NonStop Kernel**	**Windows NT Server**
Manage sourcecode	PC	PC-based source management software	PC-based source management software
	Guardian	NonStop Kernel source management software	na
	OSS	NonStop Kernel source management software	na

Both NonStop TUXEDO and NonStop SQL/MX support Micro Focus Visual COBOL. If you have a COBOL application you wish to migrate from another platform to Windows NT Server, or if you have pieces of a COBOL application that you wish to develop on Windows NT Server, be sure to check on the availability of COBOL support.

UNIX Applications

See *"Moving UNIX Applications to Windows NT Server,"* available from Microsoft Developer Network (http://www.microsoft.com/msdn/), for in-depth instructions for migrating UNIX server applications to the Windows NT Server system platform.

Client Application Development Issues

Client applications can be developed on a number of platforms, including Windows, UNIX workstations, and Macintosh. Windows is the most prevalent client with the largest installed base, and includes Windows 95, Windows NT Workstation, Windows for Workgroups, and Windows 3.11. Windows 95 and Windows NT Workstation are 32-bit operating systems, while Windows 3.11 and Windows for Workgroups (essentially the same as Windows 3.11 but with built-in networking) are operating environments that run on top of Microsoft DOS (disk operating system).

New development should take place on Windows 95 or Windows NT Workstation clients. If you have a 16-bit Windows application, you can port the application to the Win32 API. See the Microsoft Developer Network (http://www.microsoft.com/msdn/) for more information.

Windows-based Client Applications

For client development you have a great deal of flexibility in your choice of tools. Tandem recommends using the MFC (Microsoft Foundation Class) (the Win32 API specific class library for presentation and GUI) as much as possible because the applications developed will more easily port to other platforms (specifically, Macintosh and UNIX).

If you need to use an operating system service which is not part of the MFC framework, you can call the Win32 API function directly.

Use RogueWave for other class libraries if you plan to port to other platforms.

Macintosh or UNIX Clients

If you will be developing Macintosh clients in addition to Windows-based client applications, be sure to refer to Microsoft's technical publication entitled "From One Code Base to Many Platforms Using Visual C++," available from Microsoft on the Microsoft Developer Network Library CD-ROM series. Microsoft Visual C++ Cross-Development Edition for Macintosh is a set of Windows NT (or Windows 95) hosted tools for recompiling Windows code for Motorola 680x0 processors or PowerPC processors, and a library that implements Windows on the Macintosh. The Visual C++ for Macintosh product enables developers to create GUI client applications from a single source code base written to the Win32 API.

If you'll be developing for UNIX clients in addition to Windows-based clients, be sure to refer to Mainsoft Corporation and Bristol Technology for information about their tools for recompiling Win32- or MFC-based applications for UNIX systems. Using a product like Bristol Technology's Wind/U cross-platform development tool, you can maintain a single version of application source code for Windows, UNIX, OpenVMS, and OS/390 GUI. Wind/U supports the Microsoft Win32 API and MFC on the X Window system and Motif, the standard graphical environments used on UNIX, Open VMS, and OS/390. For more information, refer to Bristol Technology's Web site (http://www.bristol.com).

5 NonStop SQL/MX Solutions

The NonStop SQL/MX for Windows NT Server software product (hereafter, called simply “NonStop SQL/MX”) will enable enterprise application developers to create solutions targeted at two key strategic application areas: clustered transaction processing systems and clustered decision support systems (DSS). This section provides an overview of database applications using the NonStop SQL/MX software, with particular emphasis on DSS features and capabilities. (Transaction processing is covered in Section 6, NonStop TUXEDO Solutions.) The section also includes information about the two primary interfaces to the NonStop SQL/MX data, ISO/ANSI-92 SQL and ODBC.

DSS applications, which encompass data warehouses, data marts, and data mining applications, are strategic business solutions for the 1990s and beyond. DSS enables companies to use their vast collections of information to perform a wide range of business analytical functions, such as developing highly focused marketing strategies; detecting fraud; analyzing retail factors; managing customer life-cycle factors; and managing risk. IDC (International Data Corporation, Framingham, MA) estimates that the average return-on-investment for data warehouse implementations is more than 400%, and leading consultancies expect the DSS market to reach US $8 billion by 1999.

At the heart of any commercial-grade DSS solution is a database management system that delivers a parallel-query-based infrastructure; supports object and relational data; and provides advanced algorithms for enhanced query and data-mining functionality. The NonStop SQL/MX software product provides these features and more.

The NonStop SQL/MX Product

The NonStop SQL/MX product is a platform-independent, open relational database management system (RDBMS) that supports the ISO/ANSI-92 SQL and ODBC (open database connectivity) interfaces. In addition to support for these industry standards, the NonStop SQL/MX software includes advanced data mining, object handling, and query optimization technologies, and brings the Tandem fundamentals of high availability and scalability for the first time to the Windows NT Server platform.

Tandem demonstrated the power of RDBMSs on Windows NT Server clusters in May 1997, in a complex retail data warehouse system based on Dayton Hudson’s data warehouse, which manages retail outlets such as Target and Mervyn’s stores. This system is a 2-terabyte, 30-billion-row database deployed across a Tandem ServerNet® technology-interconnected, 64-processor Windows NT Server cluster with a single, easily managed application image. (For more information about this demonstration, referred to industry-wide as Tandem’s “2T on NT” or “2Ton” demo, see Tandem’s technical brief entitled “NonStop SQL/MX Demonstration: Two-Terabyte Database on Windows NT Server,” available on Tandem’s Web site at http://www.tandem.com/BRFS_WPS/SMX2TBBF/SMX2TBBF.htm.)

This section highlights some of the distinguishing architectural features of the NonStop SQL/MX product, many of which are features that enabled the success of the 2Ton demonstration. For a complete description of the NonStop SQL/MX software, see the

NonStop SQL/MX Database Product Description, available from Tandem's NonStop Software Web site (http://www.nonstopsw.com).

Application Programming Interfaces

The NonStop SQL/MX product conforms to the ISO/ANSI-92 SQL standard. The SQL language is the industry standard database language that provides the language elements and rules of syntax for creating, manipulating, examining, and managing information contained in relational database management systems. The three main types of functions provided by the SQL database language are described as:

- Data definition language (DDL), which provides language statements needed to define databases, define tables, define views, define integrity constraints, and define authorization privileges
- Data manipulation language (DML), which provides language statements needed to retrieve or modify (insert, delete, or update, for example) data in a database
- Data control language (DCL), which provides statements, verbs, and clauses to control the database, provide direction to the language processor, and other functions

This SQL language interface conforms to the Entry Level of the ISO/ANSI-92 SQL standard, and includes support for selected features from the Transitional, Intermediate, and Full Levels of the ISO/ANSI-92 SQL standard. For summary information about the ISO/ANSI-92 SQL levels supported in the NonStop SQL/MX software, see Table 5-3 on page 5-16; for more detailed information, see the *NonStop SQL/MX Reference Manual*, available on Tandem's Developer Page. (From Tandem's NonStop Software home page (http://www.nonstopsw.com), select "Developers' Page," and then "Manuals." High-level information about using the SQL language in the context of application programming is contained in Embedded SQL on page 5-9.

In addition, the NonStop SQL/MX product supports the ODBC 2.1 standard. ODBC (Open Database Connectivity) is an industry-standard call-level interface that aligns with the X/Open and ISO SQL language standards. ODBC is designed to enable interoperability and portability of application source code. Using ODBC is a matter of installing the appropriate driver for the database management system. For summary information about the ODBC functions, see ODBC Function Summary on page 5-24. Both high-level and detailed information about using the SQL language in the context of application programming is contained in the Microsoft ODBC programmers reference books, available from Microsoft Press, or on the Microsoft Developers Network Web site (http://www.microsoft.com/msdn/).

The next subsection describes some of the technical underpinnings to the NonStop SQL/MX software product in more detail.

NonStop SQL/MX Architecture

The NonStop SQL/MX database product consists of several components and subsystems, including the core relational database management system (the database "engine") and several underlying components from the NonStop Software infrastructure layer that are integrated with the NonStop SQL/MX database.

This subsection discusses some of the key NonStop SQL/MX database system components: specifically, the compiler, optimizer, executor, and data access manager. These components are newly designed by Tandem, implementing object-oriented design techniques, and written using Microsoft Visual C++, which means that new features and capabilities will be easy to add as new technologies, data formats, and requirements emerge.

In addition, many of the NonStop SQL/MX software components, including the executor, have been implemented using the "data-flow architecture" in which database operations (such as join, sort, and scan, for example) comprise separate objects. These objects (or operators) are linked together by input and output queues. Each operator is distinct from the others and has no dependencies upon the others. Thus, new operators can be added without affecting existing operators, and the query execution plan is simply an arrangement of these operators that most effectively handles the query.

As shown in Figure 5-1, NonStop SQL/MX Key Components, on page 5-4, the NonStop SQL/MX product is designed to run on Windows NT Server clusters[1]. A four-node Windows NT Server cluster is shown in the figure. The representative components show how the processing of a single query might be distributed by the NonStop SQL/MX components over the four nodes that comprise the cluster. In greatly simplified terms, here's how the NonStop SQL/MX components work:

- The compiler and optimizer create the data access plan. Using a unique algorithm that combines rule-based and cost-based optimization techniques, the optimizer creates the best execution plan for a given query.

  The optimizer can also instruct the executor to implement acceleration techniques, such as hash algorithms to support parallel processing without sorting, for example.

- The plan is stored as an SQL object in the database, where it will be referred to again for subsequent iterations of the same query.

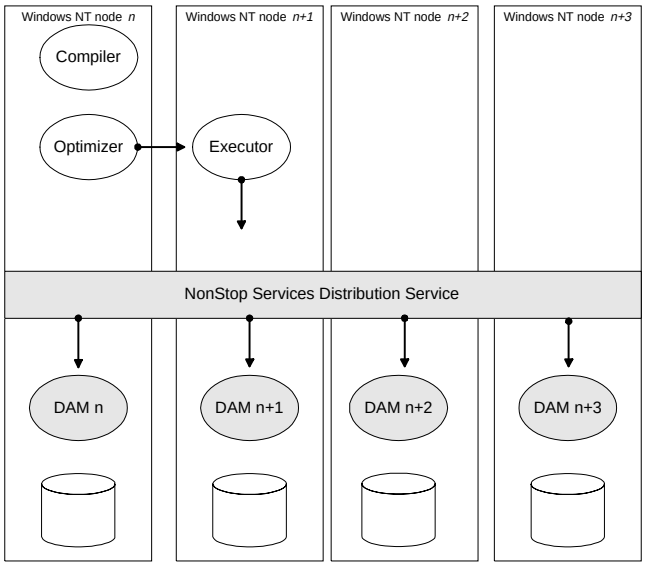
- The executor runs the instructions contained in the optimized plan. The executor distributes processing across nodes, depending upon how the database is partitioned. The executor is a multithreaded process, so it can work on multiple SQL statements and on multiple execution tree nodes at the same time.

- The data access manager (DAM) controls access to the disk and performs such tasks as reading data, updating data, and performing low-level work, including managing the cache and locks on the data located on a particular node. In addition, rather than perform the SQL operations elsewhere in the database engine, the data access manager performs the SQL operations (SELECT, ADD, COUNT, SUM, and JOIN, for example) directly on data as it comes off the disk, so that only relevant data is sent to the executor.

  As shown in Figure 5-1 and Figure 5-2, each node in the cluster has its own data access manager.

1. The NonStop SQL/MX software product is also supported on Windows NT Server systems (SMP models) and will be supported on NonStop Himalaya servers.

Figure 5-1. NonStop SQL/MX Key Components

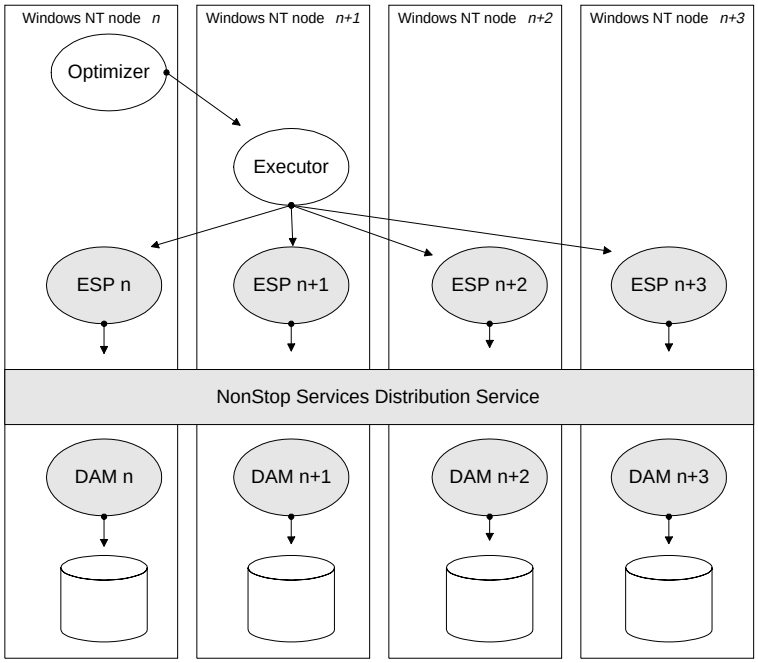


This figure shows an example of a four-node Windows NT Server cluster. Each node has its own data access manager (DAM). As its name implies, the DAM manages access to data on the disk, including managing the cache and locks on the data located on a particular node, and performs other low-level tasks such as reading and updating data.

The NonStop SQL/MX software product automatically divides a query into subtasks that execute in parallel (without special coding). The greater the number of processors or nodes in a cluster, the greater the level of parallelism and the faster the response time (see Figure 5-2). All major query tasks, including table scans, joins, sorts, and index creation, can be divided into multiple subtasks and processed in parallel.

Furthermore, based on the number and size of physical partitions and the amount of processing that needs to be completed, multiple executor server processes (ESPs) per node will be started to provide fast response time. Increasing the number of ESPs per node increases the amount of I/O parallelism. (Note that although Figure 5-2 shows one ESP per node, more per node are possible.)

Figure 5-2. Tandem NonStop SQL/MX Starts Multiple ESPs for Faster Processing



Large SQL tasks (such as queries or batch updates) can be divided into subtasks and executed in parallel. In addition to processing the subtasks on multiple nodes of a cluster, in an SMP environment (as opposed to clustered), the NonStop SQL/MX executor can process subtasks as threads.

NonStop Services

In addition to the design of the core relational database management system, additional value derives from the NonStop Services infrastructure layer with which the NonStop SQL/MX product is integrated.

The NonStop Services infrastructure layer includes components such as the NonStop Services Transaction Distributor, NonStop Services Transaction Manager, and NonStop Services Distribution Service (the latter is shown in Figure 5-1 and Figure 5-2). In addition to providing access to the data access managers, the distribution service initiates the logging process by starting the transaction manager. For more information about the NonStop Services layer, see Tandem's NonStop Services Layer on page 2-9.

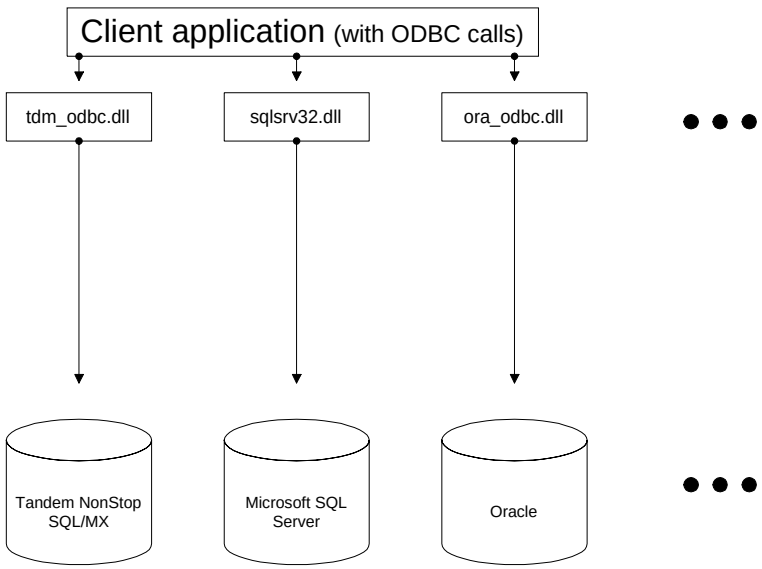
Application Architectures

Decision support (DSS) applications are inherently different from online transaction processing (OLTP) applications[1]. Within the context of DSS itself, there are also several key distinguishing application architectures. The range of solutions includes data warehouses, distributed data marts, and decision support systems built for specific end-users (executives, analysts, clerical staff, and so on). Solutions can be implemented using a two-tier or three-tier architecture, depending upon the requirements of the system.

Two-Tier Database Architectures

DSS is most frequently associated with ODBC-based query tools that work with a wide range of merchant databases. Providing that there's an ODBC driver (on Windows client workstations, the ODBC driver is typically a dynamic link library (DLL) file the end user selects via the ODBC manager control panel) to the target database, there may be little "application development" as such. This type of two-tier solution, typified in Figure 5-3, is common, and there are numerous (probably many hundreds of) data querying tools, such as those listed in Table 5-16 on page 5-28, that support the ODBC client interface.

Figure 5-3. Two-Tier Database Architecture



In a two-tier database architecture, an ODBC driver installed on the client workstation supports the data access to the target database.

In many cases, this type of two-tier model may be appropriate for DSS[2] applications. In a two-tier database architecture, an ODBC driver installed on the client workstation

1. See Table B-1, Typical DSS and OLTP Application Characteristics, on page B-1 for more information.
2. Generally speaking, for OLTP applications, Tandem recommends a three-tier architecture.

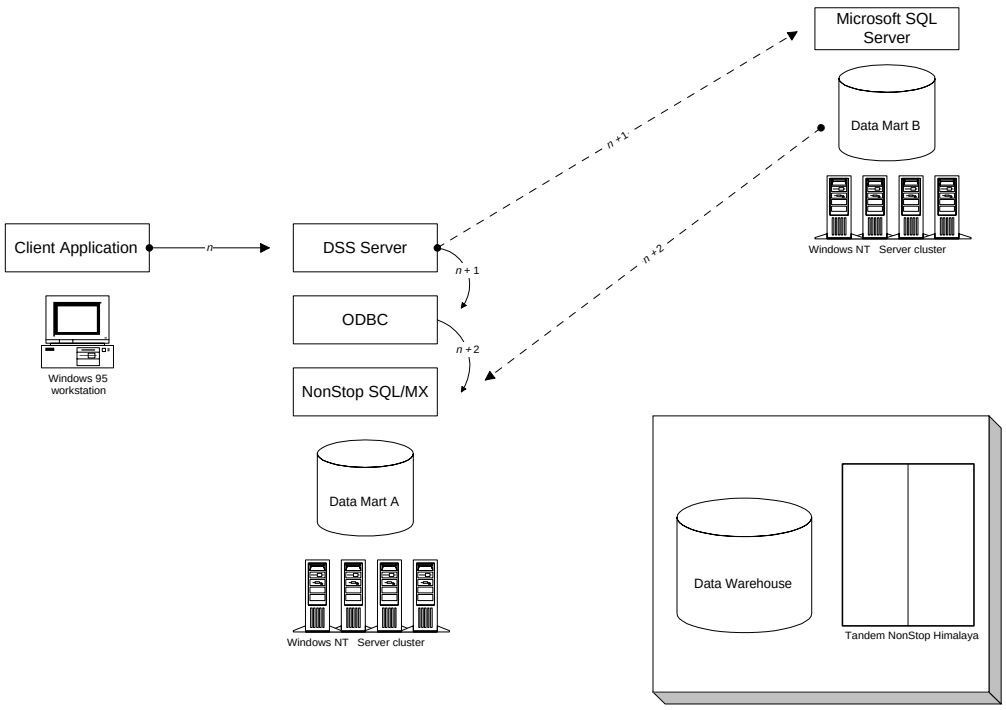
supports the data access to the target database. Vendors of database systems, Tandem included, must provide the dynamic-link library (DLL) file or other appropriate ODBC driver file for the client hardware platform, for their database systems.

Three-Tier Distributed Database Architectures

A two-tier ODBC query database solution may require little in the way of application development, but there are other types of DSS solutions that can make use of the three-tier architecture. DSS has become a strategic business application and as such is beginning to have the requirements for scalability and high availability that have historically distinguished OLTP. This means that an organization may not wish to take a DSS application off-line to reload data, but rather let users run queries against production databases. Also, many organizations are beginning to provide some access to DSS applications over the Internet, to hundreds or thousands of users.

These requirements make the case for creating a middle-tier application that contains the business and processing logic and that can support large numbers of users in realtime — the same requirements as those for OLTP.

Figure 5-4. Three-Tier Database Architecture



Three-tier client/server database applications may span heterogeneous databases and may span multiple services and servers.

Creating a data warehouse, distributed data mart, or distributed decision-support solution might involve significant application programming — for example, to pre-summarize data extracted from multiple sources and then consolidate the data in a

central database or to distribute data to multiple local databases. Building distributed database solutions involves major architectural considerations, including whether and how to partition the database tables; how frequently, if at all, to extract and load data; and whether or not some (or all) data should be replicated to local data marts. Data from operational databases might be extracted using embedded SQL, ODBC, or third-party tools. Applications such as these would include a middle-tier application containing the application processing logic, including business rules — extracting all data from three different customer information databases, for example, for accounts with balances over $5,000.

For more information about DSS and related technologies, see Tandem's DSS Web site: from the Tandem Web page at http://www.tandem.com, select "Solutions" and then select "Decision Support." See Database Tool Reference on page 5-28 for listings of some third-party data warehouse tools vendors.

Overview of Application Development

There are several ways of interacting with NonStop SQL/MX data, including:

- Embedded SQL
- ODBC
- SQLCI

These are discussed briefly in the next subsections. (Note that for the beta release of the NonStop SQL/MX product, access to the program is enabled via the Tandem_ID — which is set up during NonStop SQL/MX installation — and the Tandem_ID is mapped to the Windows NT Server Domain Administrator user identity. This means you must be on the list of domain administrators to use the product (beta release only). Be sure to check the NonStop SQL/MX (Beta release) Readme file and the *NonStop SQL/MX Installation Guide* for more information.)

SQLCI

SQLCI (conversational interface) is an interactive interface to the NonStop SQL/MX database that supports SQL commands, statements, and ad hoc queries through a console-based GUI interface. SQLCI is intended primarily for system administrators or database administrators (DBAs) to interact with the database, but developers may also wish to use SQLCI to test queries or other SQL statements that they're planning to use in SQL applications, because SQLCI supports SQL statements in addition to SQLCI commands.

As an example, the SQLCI DISPLAY_EXPLAIN command displays a statement's execution plan. A developer might want to use this command to compare execution plans for alternative SQL statements before embedding them in source code, and then choose one SQL statement over the other.

See the *NonStop SQL/MX Reference Manual*, available on the NonStop Software Developer's Page for additional information about SQLCI. From Tandem's NonStop Software home page (http://www.nonstopsw.com), select "Developers' Page," and then "Manuals."

Embedded SQL

The NonStop SQL/MX database is ISO/ANSI-92 SQL compliant. As such, it supports SQL statements and directives embedded in a host language. The ISO/ANSI-92 SQL standard provides support for seven host languages. Of these seven languages, C and COBOL are currently supported by the NonStop SQL/MX product; support for C++ will be added in future releases.

The fact that the NonStop SQL/MX database is ISO/ANSI-92 compliant means that you can embed standard ISO/ANSI-92 SQL directly in your application: the SQL statements are coded in advance and compiled into the application; the NonStop SQL/MX optimizer generates an efficient path for fulfilling SQL statements, and stores this information in the database for use by all such future requests (as discussed in NonStop SQL/MX Architecture on page 5-2).

When the application executes, the SQL statements within the application access the data and pass it to the host language for processing. For detailed information about which SQL statements and directives can be embedded in C/C++ or COBOL programs, see the "*NonStop SQL/MX Programming Manual for C and COBOL,*" available on the NonStop Software Developers' Page and the NonStop SQL/MX installation CD.

Embedded SQL statements and any variables that will be shared by the host language and the embedded SQL statements must be identified by use of syntax markers appropriate to the specific host language. In the supported host languages, each embedded SQL statement (or directive to the SQL language processor) must begin with the words "EXEC SQL".

The terminator is specific to the host language. For example, the COBOL85 syntax requires the words "END-EXEC." to mark the end of an SQL statement, as in:

```
EXEC SQL SQL statement or directive END-EXEC.
```

In a C program, each embedded SQL statement must end with a semicolon, as in this example:

```
EXEC SQL SQL statement or directive;
```

Before using a host variable within the context of an embedded SQL statement, you must have declared the variable in the DECLARE SECTION of your program. (As with any other embedded SQL statements, the DECLARE SECTION of the program must be flagged using the BEGIN DECLARE SECTION and END DECLARE SECTION directives.) Then, when using the variable within the context of the embedded SQL statement, you must precede the name of the variable with a colon. For example, *variable1* in the body of a C program should be listed as *:variable1* if it is also going to be used in an embedded SQL statement within the same C program. For complete details, see the *NonStop Programming Manual for C and COBOL.*

There are two types of embedded SQL: static embedded SQL and dynamic embedded SQL. Other vendor implementations of embedded SQL may present different concepts of these terms, but in this guide the discussion is limited to the NonStop SQL/MX perspective.

Static Embedded SQL

In static embedded SQL, the general format of a query, statement, or SQL directive doesn't change, although variables within the statement itself change. A static embedded SQL statement enables repeated inserts, updates, or deletes to tables in a database.

For example, a simple branch banking application may have a client interface that enables loan officers to review account balances from their PCs. A simple data entry dialog box might allow users to enter an account number or name only, which is then submitted to the database as a query to return the customer's account balances. The query is static in the sense that the structure of the SQL statement is consistent, again and again. The account number is entered as a variable, which gets inserted in the same place each time within the embedded SQL statement.

Dynamic Embedded SQL

Dynamic embedded SQL differs from static embedded SQL in that the entire SQL statement (or directive) is itself an unknown entity until it is submitted. Thus, the entire SQL statement is a variable: a character string variable takes the place of the entire SQL statement, not just a variable within the statement, and the statement is neither formed nor compiled until runtime.

The branch banking application described above, for example, might be modified to provide a flexible tool that enables loan officers to perform a wide range of activities on customer accounts beyond querying account balances, including updating customer information or overriding balances, by incorporating dynamic embedded SQL statements rather than static SQL statements. The embedded SQL statement may be an unknown entity until the moment the loan officer is performing the activity.

Some of the key characteristics of dynamic and static embedded SQL are summarized in Table 5-1.

Table 5-1. Dynamic SQL and Static SQL Characteristics Compared

Static SQL	Dynamic SQL
DECLARE SECTION required for any variables to be shared between host language and SQL statements	Uses SQLDA (Descriptor Area) for dynamic number of parameter
Uses host variables (:variable1, :variable2)	Uses SQL parameters (?,?)
Access plan is precompiled and stored in database as an object	Access plan developed at runtime
Specification and execution in the same line	Uses two separate statements (PREPARE and EXECUTE) for declaration and execution
Precompilation process necessary	Precompilation process not necessary
Scrollable cursors in ISO/ANSI-92 SQL	Extended dynamic cursors (ALLOCATE CURSOR)

ODBC

In addition to using ODBC in the context of a two-tier solution, (as described in Two-Tier Database Architectures on page 5-6), you can use ODBC in the middle tier of a three-tier client/server application. ODBC is a call-level interface, and so rather than embedding SQL (or, in addition to embedding SQL), you can make calls from within the server side of the application using ODBC. When the application targets more than one database, or if the language being used to develop the server (middle-tier) code doesn't directly support embedded SQL calls — Visual Basic, for example — you can use ODBC calls in the server code. Microsoft has been encouraging this development model for some time now; the result of using ODBC in this way is that your code is more portable.

So although the NonStop SQL/MX software is ISO/ANSI-92 SQL compliant, the SQL compiler is specific to Tandem NonStop SQL/MX (as is any other vendor's SQL precompiler specific to its internal database language). This means that to deploy a NonStop SQL/MX application with a competing database management system, you would have run the source code again through the SQL precompiler for the target database, and go through the whole compile and link process again.

Using ODBC rather than embedded SQL enables you to integrate a new database with the application, by implementing the appropriate ODBC driver for the database.

ODBC is a version of the SQL call-level interface (CLI) that specifies additional functionality to the SQL CLI, including integration with Microsoft Windows environment. The ODBC CLI performs the following functions:

- Provides a locally bound call interface
- Instantiates a session with a named data source
- Defines local storage
- Sends SQL requests to that data source
- Retrieves results data produced by that data source

Application Development Process

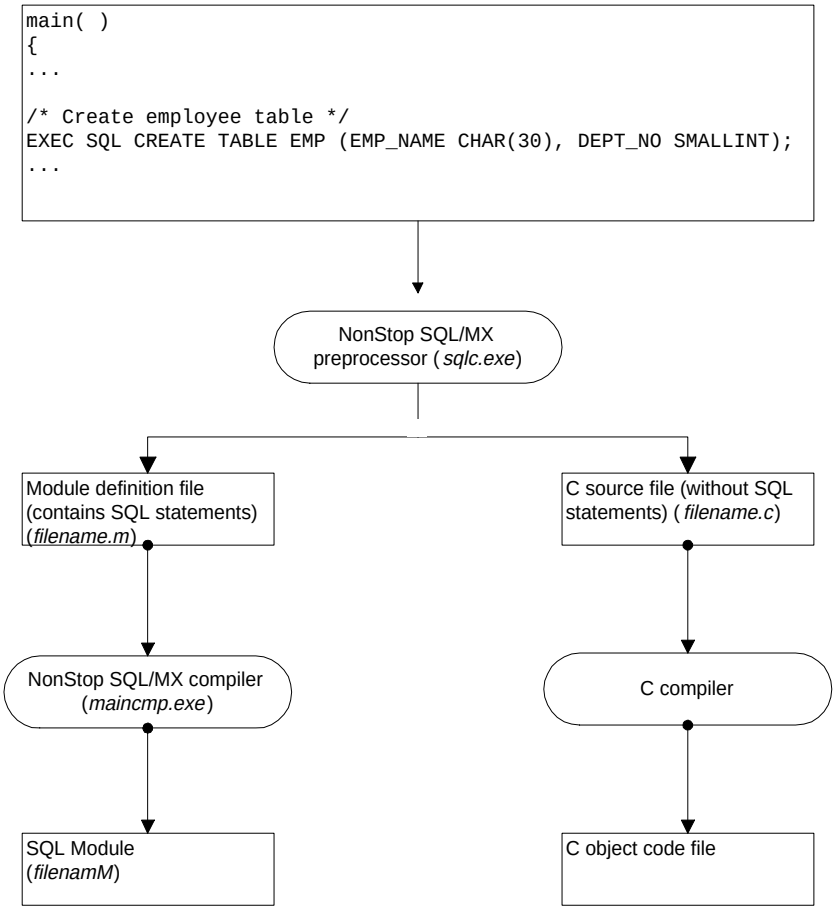
The NonStop SQL/MX software product includes a host-language-specific preprocessor for preprocessing application source code in C or COBOL that contains SQL statements embedded in the text of the source file. You can use the C or COBOL compiler of your choice as long as it's supported on the Windows NT Server platform, but note that the Microsoft Windows Visual C++ environment is recommended for C/C++ application development and Micro Focus Visual COBOL is recommended for COBOL applications.

Figure 5-5, NonStop SQL/MX Preprocessor-Compiler Overview, provides an overview of the process. Here are some implementation details:

For the preliminary release of the NonStop SQL/MX software product, you must have Domain Administrator privileges (a Tandem_ID gets created and mapped to a Windows NT user account having these privileges) to use the preprocessor and other NonStop SQL/MX related features and functions.

Include the sqlcli.h file (located in the /cli subdirectory) in any C program that contains embedded SQL. This is the SQL/MX call-level interface (CLI) header file, which is needed by all programs containing embedded SQL; the preprocessor (sqlc for C/C++ and sqlco for COBOL) must find this file when it preprocesses your source file.

Figure 5-5. NonStop SQL/MX Preprocessor-Compiler Overview



The following list of steps summarizes the application development process for an application written in standard C for use with the NonStop SQL/MX database.

1. Write the program logic for the application that will run as a Windows NT service[1] and access data contained in NonStop SQL/MX database tables. Depending upon your application design and needs, the source code file can contain:
   - Embedded static SQL statements and directives
   - Embedded dynamic SQL statements and directives
   - ODBC function calls

1. A Windows NT service is a program that runs whenever the server hardware is running the Windows NT operating system; it does not require users to be logged on in order to run, nor does it require a user interface.

- A combination of all three types of calls

2. Follow the syntax rules for the host language processor, which will typically be either Microsoft Visual C++, Microsoft Visual Basic, Borland C++, or one of the other graphical development environments targeted for Windows platforms.
3. Run the appropriate NonStop SQL/MX preprocessor for the host language (in this example, C) to process any embedded SQL statements and directives and generate:
   - The host language source file, stripped of embedded SQL statements
   - An SQL module definition file (MDF), to be compiled by the SQL compiler. The content of the module definition file will vary, depending upon whether the source file contained static or dynamic embedded SQL statements.

     By default, all the module definition files you create during your pre-processing are stored in the \TDM_SQLmodules subdirectory, unless you've set the appropriate Windows NT Server environment variable to something else.

     - For dynamic SQL statements, the MDF contains only the name of the statement, which will be stored as an object in the database.
     - For static SQL statements, the MDF contains the name of the statement; the content; the input host variables; and the output host variables.
4. Using the C compiler, compile the C source files generated by the preprocessor. The C compiler generates an object file that contains C object code.
5. Run the SQL compiler (arkcmp) to compile the SQL source statements in the file generated by the preprocessor; this registers it as a module in the SQL catalog and validates the output SQL program for execution.
6. Install the executable file as a service under the Windows NT Server operating system.

Sample Application

See Tandem NonStop Software Developers' Page for more information about NonStop SQL/MX DSS application development. From Tandem's NonStop Software home page (http://www.nonstopsw.com), select "Developers' Page." The *NonStop SQL/MX Programming Manual for C*, available from the Developers' Page, contains sample databases and sample applications.

Recommendations

The NonStop SQL/MX product will be released in several stages, with increasing levels of functionality. Depending upon the functionality that you wish to include in your application, you may need to comment out lines of code before compiling until those features are supported. See NonStop Software for Windows NT Server Features on page 3-15 for details about functionality and availability in the NonStop SQL/MX product.

If you are an application developer writing server-based solutions, use C or COBOL85 as the host language, and use embedded SQL or ODBC to access NonStop SQL/MX

data. To keep your application as portable and open as possible, avoid using any extensions.

Also, avoid calls that directly access the operating system. Use the NonStop SQL/MX and NonStop TUXEDO products as an open middleware, abstraction layer. Isolate security, error handling, catalog access, and other routines that might comprise a framework into appropriate code modules.

For NonStop SQL/MP Developers

Developers who are already familiar with Tandem's NonStop SQL/MP product should be aware that the NonStop SQL/MX product is a completely redesigned database management system. The ISO/ANSI-92 SQL support provides many features not found in NonStop SQL/MP, including features such as:

- Naming
- Grant/Revoke
- Referential integrity
- Indexes with NULL objects
- UNION operations in views
- CAST, a function that converts one datatype to another for use in a formula, for example
- FROM clause with SELECT statement

In addition, the database engine has an improved optimizer technology, improved usage of executor server processes (ESPs), a catalog that is better for large tables with many partitions, a call-level interface in addition to embedded SQL for application coding, different ways to manage SQL modules used by programs which will enable sharing of SQL code programming between programs and easier management of SQL code changes.

Note also that the preprocessor is now a separate function. Under NonStop SQL/MP, the preprocessor was part of the SQL compiler, but that function has now been separated from the compiler.

Many Tandem extensions from NonStop SQL/MP are not supported in the NonStop SQL/MX product. An example of how the changes may affect your coding is that because table and view names are handled differently in the NonStop SQL/MX product, the concept of the Tandem extension DEFINE is no longer relevant (nor is it supported), therefore, you'll be handling names differently in your application code. Refer to Differences Between NonStop SQL/MP and NonStop SQL/MXof 3) on page 5-18 for summary information.

API Reference

There are two primary interfaces supported for server-based applications: ISO/ANSI-92 SQL and ODBC.

ISO/ANSI-92 SQL

The ISO/ANSI-92 SQL standard defines three conformance levels: Entry, Intermediate, and Full. The United States federal government added a fourth level between the Entry and Intermediate, known as Transitional. The initial release of the NonStop SQL/MX product complies with the Entry level[1] and will provide most all (21 out of 24) of the features from the Transitional level. (The only Transitional level features NonStop SQL/MX does not support are *information schema views* and *nullable primary keys.*) In addition, several features of the Intermediate and Full levels of the ISO/ANSI-92 SQL standard are supported. These features are summarized in Table 5-2.

Table 5-2. ISO/ANSI-92 SQL Features in NonStop SQL/MX (page 1 of 2)

Level	Feature
Entry	ANSI Names
	ANSI Schema
	ANSI Security (GRANT)
	ANSI Views
	Referential Integrity
	Statement Atomicity (will be supported in a future release)
	String Functions
	Unique Constraints
	Updatable Primary Keys
Intermediate	Datetime
	Dynamic SQL
	Get Diagnostic Statement
	Left Outer Join
	Natural Join
	Referential Delete Actions
	Revoke (no Cascade, modified Restrict)
	Right Outer Join
	Union in Views
	Dynamic SQL
	Revoke (no Cascade, modified Restrict)

1. Statement atomicity will be supported in a future release.

Table 5-2. ISO/ANSI-92 SQL Features in NonStop SQL/MX (page 2 of 2)

Level	Feature
Intermediate	Drop Schema
	Drop Table
Full	Table Expression

This subsection includes summaries of supported level features, as well as a summary of the differences between Tandem NonStop SQL/MP and NonStop SQL/MX. For complete details of the NonStop SQL/MX language elements and usage, see the *NonStop SQL/MX Reference Manual*.

Table 5-3. ISO/ANSI-92 SQL Levels Supported in NonStop SQL/MX (page 1 of 2)

Level	Features
Entry	Upper-case identifiers only
	Referential integrity
	ANSI Names, ANSI Views, ANSI Schema, ANSI Security (GRANT), Updateable primary keys
	String functions
	Unique contraints
Transitional	Case-insensitive identifiers
	DATETIME data types
	VARCHAR data type
	NATURAL JOIN, INNER JOIN, LEFT OUTER JOIN and RIGHT OUTER JOIN clauses
	CHAR_LENGTH, SUBSTRING, and TRIM functions and concatenation operator on character data types
	CAST clause (Implicit casting between different numeric types; Implicit casting between different character data types)
	UNION in view definitions
	Referential delete actions
	Transaction access modes, transaction isolation levels and SET TRANSACTION statement
	Dynamic SQL
	Diagnostics area and GET DIAGNOSTICS statement
Intermediate	User-defined constraint names
	Scalar subqueries

Table 5-3. ISO/ANSI-92 SQL Levels Supported in NonStop SQL/MX (page 2 of 2)

Level	Features
	Relax Entry SQL restriction on LIKE and NULL predicates
	Relax Entry SQL restriction on local variable scoping in embedded SQL programs
	Relax Entry SQL restriction on UNION operator and query expressions
	Relax Entry SQL restriction on row value constructors
Full	Arbitrary precision for seconds
	Allow derived tables in FROM clause
	Relax Intermediate SQL restriction on underscore character in identifiers
	Relax Intermediate SQL restriction on data types of indicator parameters and indicator variables
	Relax Intermediate SQL restriction on value expressions
	Relax Intermediate SQL restriction on the use of DISTINCT keyword
	Relax Intermediate SQL restriction on the order of column names in a referential constraint definition

Functional Comparison: NonStop SQL/MX to NonStop SQL/MP

This subsection summarizes some of the major functional differences between NonStop SQL/MX and Tandem's earlier database product, NonStop SQL/MP. Developers who are familiar with NonStop SQL/MP will notice some major differences between NonStop SQL/MX and NonStop SQL/MP. Most of the differences are due to the fact that NonStop SQL/MX conforms to ISO/ANSI-92 SQL.

Table 5-4. Differences Between NonStop SQL/MP and NonStop SQL/MX (page 1 of 3)

Category or Topic	NonStop SQL/MX	NonStop SQL/MP
Constraints	Allows specifying all the constraints uniformly as part of table definition itself and supports the specification of referential integrity constraints between tables.	Allows specifying non-null columns and primary keys as part of table definition, checking constraints through CREATE CONSTRAINT statements and unique columns through CREATE UNIQUE INDEX statements.
Datatype support	Datetime datatype mapping	See Table 5-8
Directives	Supports GET DIAGNOSTICS and GET/SET DESCRIPTOR statements in place of INCLUDE SQLCA, SQLDA, and SQLSA, respectively.	Supports INCLUDE SQLCA, INCLUDE SQLDA, and INCLUDE SQLSA.
Error code format	Supports SQLSTATE as well as SQLCODE mechanism.	Provides an integer-valued mechanism, SQLCODE, for exchanging error information between the database and application.
Identifier formats	Supports both *regular identifiers* and *delimited identifiers* (which are formed using any character in the character set supported by the implementation, but enclosed in double quotation marks).	Supports only *regular identifiers* (which are formed out of alphanumeric and underscore characters).
Internationalization	Not supported in the first release, but will support ISO/ANSI-92 SQL compliant internationalization mechanism in a future release.	Implements a number of predefined character sets, and allows users to create named collations and to associated a specific collation with a character string data type, column, and expression.

Table 5-4. Differences Between NonStop SQL/MP and NonStop SQL/MX (page 2 of 3)

Category or Topic	NonStop SQL/MX	NonStop SQL/MP
Metadata organization	Objects described in base tables that comprise *schema metadata dictionary* associated with each schema. Selected metadata information is replicated in corresponding file labels and resource forks.	Objects described in base tables that comprise *catalogs* (Guardian disk subvolumes).
Names of tables, views, and other database objects	Purely logical names with their own namespace.	Correspond directly to Guardian file names and share Guardian file namespace.
Namespace	Object names qualified by schema and catalog names *(schema.catalog.name)* (The catalog in NonStop SQL/MX is not the same concept as in NonStop SQL/MP.)	Object names qualified by Guardian disk volume and subvolume names *($vol5.sales.parts)*
Orthogonality of the SQL interface	Allows arbitrary queries wherever table names are allowed.	Does not allow arbitrary queries wherever table names are allowed.
Primary key updates	Allowed.	Not allowed.
Security	Database objects are secured by means of *privileges*, which can be *granted* and *revoked.*	Database objects are secured using the Guardian file access control options (in the same way Guardian files are secured).

Table 5-4. Differences Between NonStop SQL/MP and NonStop SQL/MX (page 3 of 3)

Category or Topic	NonStop SQL/MX	NonStop SQL/MP
String delimiters	Single quotation marks to enclose literals.	Allows use of double quotes to enclose literals.
Transactions	Allows specifying the isolation level at both statement and transaction levels.	Allows specifying *access modes* (*isolation leve*l in ISO/ANSI-92 SQL terminology) for each statement, while prohibiting the specification at transaction level.
	Does not guarantee statement atomicity.	Guarantees statement atomicity.
	Different statement level access option.	
Views	Supports independently securable updating only and depends solely on the query expression that defines the view.	Supports two classes of views, *protection views* and *shorthand views*. Protection views can be updated and users can control the security rights on the views. Security rights on shorthand views are not independently controllable.

Table 5-5. Datetime Data Type Mapping

NonStop SQL/MX	NonStop SQL/MP
DATE	DATE
DATE	DATETIME YEAR TO DAY
TIME	TIME
TIME	DATETIME HOUR TO SECOND
TIME (f)	DATETIME HOUR TO FRACTION
TIMESTAMP	TIMESTAMP
TIMESTAMP(0)	DATETIME YEAR TO SECOND
TIMESTAMP(f)	DATETIME YEAR TO FRACTION(f)
Not supported	Other datetime ranges such as YEAR TO MONTH

Table 5-6. Interval Data Type Mapping (page 1 of 2)

NonStop SQL/MX	NonStop SQL/MP
INTERVAL YEAR (n)	INTERVAL YEAR (n)
INTERVAL YEAR (n) TO MONTH	INTERVAL YEAR (n) TO MONTH
INTERVAL MONTH (n)	INTERVAL MONTH (n)
INTERVAL DAY (n)	INTERVAL DAY (n)
INTERVAL DAY (n) TO HOUR	INTERVAL DAY (n) TO HOUR
INTERVAL DAY (n) TO MINUTE	INTERVAL DAY (n) TO MINUTE
INTERVAL DAY (n) TO SECOND (0)	INTERVAL DAY (n) TO SECOND (0)
INTERVAL DAY (n) TO SECOND (f)	INTERVAL DAY (n) TO FRACTION (f)
INTERVAL HOUR (n)	INTERVAL HOUR (n)
INTERVAL HOUR (n) TO MINUTE	INTERVAL HOUR (n) TO MINUTE
INTERVAL HOUR (n) TO SECOND (0)	INTERVAL HOUR (n) TO SECOND
INTERVAL HOUR (n) TO SECOND (f)	INTERVAL HOUR (n) TO FRACTION (f)
INTERVAL MINUTE (n)	INTERVAL MINUTE (n)
INTERVAL MINUTE (n) TO SECOND (0)	INTERVAL MINUTE (n) TO SECOND
INTERVAL MINUTE (n) TO SECOND (f)	INTERVAL MINUTE (n) TO FRACTION (f)

Table 5-6. Interval Data Type Mapping (page 2 of 2)

NonStop SQL/MX	NonStop SQL/MP
INTERVAL SECOND (n, 0)	INTERVAL SECOND (n)
INTERVAL SECOND (n, f)	INTERVAL SECOND (n) TO FRACTION (f)
Not supported	INTERVAL FRACTION (n) TO FRACTION (f)

Language Comparison: NonStop SQL/MX to NonStop SQL/MP

This subsection summarizes the differences between the SQL language supported by NonStop SQL/MX and that supported by NonStop SQL/MP. Developers already familiar with NonStop SQL/MP should read this subsection to be aware of the differences between the two products.

New Statements (Supported in NonStop SQL/MX; Not Supported in NonStop SQL/MP)

ALLOCATE CURSOR

ALLOCATE DESCRIPTOR

ALTER TABLE ADD CONSTRAINT

ALTER TABLE DROP CONSTRAINT

ALTER TABLE SET CONSTRAINT

CREATE SCHEMA

DEALLOCATE DESCRIPTOR

DEALLOCATE PREPARE

DECLARE CATALOG

DECLARE SCHEMA

DROP SCHEMA

GET DESCRIPTOR

GET DIAGNOSTICS

GRANT

REPLICATE CATALOG

REVOKE

SET CATALOG

SET DESCRIPTOR

SET SCHEMA

SET TRANSACTION

Deprecated Statements (Supported in NonStop SQL/MP, Not Supported in NonStop SQL/MX)

ALTER CATALOG

ALTER COLLATION

ALTER INDEX SECURE

ALTER PROGRAM

ALTER TABLE OWNER

ALTER TABLE SECURE

ALTER TABLE SIMILARITY CHECK

ALTER VIEW OWNER

ALTER VIEW SECURE

ALTER VIEW SIMILARITY CHECK

CREATE COLLATION

CREATE CONSTRAINT

CREATE SYSTEM CATALOG

DROP COLLATION

DROP CONSTRAINT

DROP SYSTEM CATALOG

Semantical Differences (Statements that Have Different Meanings in NonStop SQL/MX and NonStop SQL/MP)

CREATE CATALOG

DROP CATALOG

ODBC Function Summary

The following tables, excerpted from the *ODBC Programmer's Guide* (available through Microsoft Developer Network Library and from Microsoft's Web site (http://www.microsoft.com), briefly describe the ODBC functions. Because not all drivers support the complete function set, and because there are several different levels of the standard to which functions can conform, it's best to include code in an application to obtain conformance information by calling the SQLGetInfo function at the beginning of ODBC code sections. In addition, to obtain information about support for a specific function in a driver, code your application to call SQLGetFunctions.

Table 5-7. Connect to a Data Source

Function Name	Purpose
SQLAllocEnv	Obtains an environment handle. One environment handle is used for one or more connections.
SQLAllocConnect	Obtains a connection handle.
SQLConnect	Connects to a specific driver by data source name, user ID, and password.
SQLDriverConnect	Connects to a specific driver by connection string or requests that the Driver Manager and driver display connection dialog boxes for the user.
SQLBrowseConnect	Returns successive levels of connection attributes and valid attribute values. When a value has been specified for each connection attribute, connects to the data source.

Table 5-8. Obtain Information about a Driver and Data Source

Function Name	Purpose
SQLDataSources	Returns the list of available data sources.
SQLDrivers	Returns the list of installed drivers and their attributes.
SQLGetInfo	Returns information about a specific driver and data source.
SQLGetFunctions	Returns supported driver functions.
SQLGetTypeInfo	Returns information about supported data types.

Table 5-9. Set and Retrieve Driver Options

Function Name	Purpose
SQLSetConnectOption	Sets a connection option.
SQLGetConnectOption	Returns the value of a connection option.
SQLSetStmtOption	Sets a statement option.
SQLGetStmtOption	Returns the value of a statement option.

Table 5-10. Prepare SQL Requests

Function Name	Purpose
SQLAllocStmt	Allocates a statement handle.
SQLPrepare	Prepares an SQL statement for later execution.
SQLBindParameter	Assigns storage for a parameter in an SQL statement.
SQLParamOptions	Specifies the use of multiple values for parameters.
SQLGetCursorName	Returns the cursor name associated with a statement handle.
SQLSetCursorName	Specifies a cursor name.
SQLSetScrollOptions	Sets options that control cursor behavior.

Table 5-11. Submit Requests (page 1 of 2)

Function Name	Purpose
SQLExecute	Executes a prepared statement.
SQLExecDirect	Executes a statement.
SQLNativeSql	Returns the text of an SQL statement as translated by the driver.
SQLDescribeParam	Returns the description for a specific parameter in a statement.

Table 5-11. Submit Requests (page 2 of 2)

SQLNumParams	Returns the number of parameters in a statement.
SQLParamData	Used in conjunction with SQLPutData to supply parameter data at execution time. (Useful for long data values.)
SQLPutData	Send part or all of a data value for a parameter. (Useful for long data values.)

Table 5-12. Retrieve Results or Retrieve Information About Results

Function Name	Purpose
SQLRowCount	Returns the number of rows affected by an insert, update, or delete request.
SQLNumResultCols	Returns the number of columns in the result set.
SQLDescribeCol	Describes a column in the result set.
SQLColAttributes	Describes attributes of a column in the result set.
SQLBindCol	Assigns storage for a result column and specifies the data type.
SQLFetch	Returns a result row.
SQLExtendedFetch	Returns multiple result rows.
SQLGetData	Returns part or all of one column of one row of a result set. (Useful for long data values.)
SQLSetPos	Positions a cursor within a fetched block of data.
SQLMoreResults	Determines whether there are more result sets available and, if so, initializes processing for the next result set.
SQLError	Returns additional error or status information.

Table 5-13. Obtain Information About Source Database Tables (Catalog Functions) (page 1 of 2)

Function Name	Purpose
SQLColumnPrivileges	Returns a list of columns and associated privileges for one or more tables.
SQLColumns	Returns the list of column names in specified tables.
SQLForeignKeys	Returns a list of column names that comprise foreign keys, if they exist for a specified table.
SQLPrimaryKeys	Returns the list of column names that comprise the primary key for a table.
SQLProcedureColumns	Returns the list of input and output parameters, as well as the columns that make up the result set for the specified procedures.

Table 5-13. Obtain Information About Source Database Tables (Catalog Functions) (page 2 of 2)

SQLProcedures	Returns the list of procedure names stored in a specific data source.
SQLSpecialColumns	Returns information about the optimal set of columns that uniquely identifies a row in a specified table, or the columns that are automatically updated when any value in the row is updated by a transaction.
SQLStatistics	Returns statistics about a single table and the list of indexes associated with the table.
SQLTablePrivileges	Returns a list of tables and the privileges associated with each table.
SQLTables	Returns the list of table names stored in a specific data source.

Table 5-14. Terminate a Statement

Function Name	Purpose
SQLFreeStmt	Ends statement processing and closes the associated cursor, discards pending results, and, optionally, frees all resources associated with the statement handle.
SQLCancel	Cancels an SQL statement.
SQLTransact	Commits or rolls back a transaction.

Table 5-15. Terminate a Connection

Function Name	Purpose
SQLDisconnect	Closes the connection.
SQLFreeConnect	Releases the connection handle.
SQLFreeEnv	Releases the environment handle.

Database Tool Reference

Many vendors specialize in providing the tools and mechanisms needed for creating, deploying, querying, or managing distributed decision support systems, including data warehouses. Some of these vendors and products are included in Table 5-16, Table 5-17, and Table 5-18. Note that these lists are simply starting points: there are likely hundreds of query tools on the market, for example. Also, many vendors may produce tools that span many different categories. The products in these lists range from ODBC-based query and reporting tools; tools for extracting data from multiple sources and loading it into the data warehouse or data mart; data mining tools; and online analytical processing (OLAP) tools.

Note that this information is being provided for developers to use as a starting point for their own research, and being listed in the table does not confer an endorsement by Tandem for any product. In addition, consult Development Tools for Distributed Applications on page 6-21 for information about development tools for three-tier client/server applications.

Table 5-16. Query, Reporting, and Other DSS Tools

Vendor	Example Product Name	Web Address
Angoss Software International Limited	KnowledgeSeeker	http://www.angoss.com
Arborsoft	EssBase Web Gateway	http://www.arborsoft.com
Applix, Inc.	RealTime, Spreadsheet	http://www.applix.com
Brio Technology	BrioQuery, Brio Enterprise, Insight	http://www.brio.com
Business Objects	WebIntelligence	http://www.businessobjects.com
Comshare, Inc.	Commander	http://www.comshare.com
Decisionmark	Proximity, Decisionware	http://www.decisionmark.com
Intersolv, Inc.	DataDirect	http://www.intersolv.com
IQ Software	IQ/LiveWeb	http://www.iqsc.com
MicroStrategy	DSSAgent, DSSServer, DSSWeb	http://www.strategy.com
Platinum Technology, Inc.	InfoPump, InfoQuery, DBVision, Forest&Trees	http://www.platinum.com
Software AG	SourcePoint	http://www.sagus.com
Speedware	Esperant, Media	http://www.speedware.com
Xense Technology	DB Publisher	http://www.xense.com

Table 5-17. Data Mining, OLAP, and Related Tools

Vendor	Example Product Name	Web Address
Andyne Computing*	GQL, QGL Reports	http://www.andyne.on.ca
Data Distilleries B.V.*	Data Surveyor	http://www.ddi.nl
DataMind Corp.	DataCruncher	http://www.datamindcorp.com
Dimensional Insight, Inc.	DI-Diver, DI-Atlantis	http://www.dimins.com
Gentia Software	WebSuite	http://www.gentia.com
Information Advantage, Inc.	WebOLAP, DecisionSuite	http://www.infoadvan.com
Information Discovery	Data Mining Suite	http://www.datamining.com
Magnify, Inc.*	PATTERN	http://www.magnify.com
Naviant Technology Solutions	IMPACT	http://www.naviant.com
Syllogic B.V.*	Adaptive Enterprise Management	http://www.syllogic.nl
NeoVista Solutions, Inc.*	Decision Series, DecisionAccess, DecisionNet	http://www.neovista.com
Seagate Software Information Management Group	Holos	http://www.seagatesoftware.com/holos/

* Tandem partner for Data Mining applications.

Table 5-18. Data Warehouse/Data Mart Development, Management, Deployment Tools (page 1 of 2)

Vendor	Example Product Name	Web Address
Carleton	PASSPORT	www.carleton.com
Cognos Inc.	PowerPlay, 4Thought	http://www.cognos.com
Cross Access Corp.	CrossAccess, CrossXpress, DataMapper	http://www.crossaccess.com
Informatica Corp.	PowerMart	http://www.informatica.com
Information Builders	SmartMart	http://www.ibi.com
Interweave Software, Inc.	Interweave Modeler	http://www.iweave.com
Pilot Software	Decision Support Suite	http://www.pilotsw.com

Table 5-18. Data Warehouse/Data Mart Development, Management, Deployment Tools (page 2 of 2)

Vendor	Example Product Name	Web Address
Prism Solutions, Inc.	PrismWeb Access, Warehouse Directory	www.prismsolutions.com
Speedware	Esperant, Media	http://www.speedware.com
Vmark Software	DataStage, UniVerse	http://www.vmark.com

6 NonStop TUXEDO Solutions

The NonStop TUXEDO for Windows NT Server product is designed to provide high-availability, scalability, and manageability to a wide array of client/server distributed applications, including:

- Web-based systems that must scale to support possibly thousands of users, whose activities may encompass multiple other servers, services, and possibly, databases. Whether you're developing a Web-enabled e-commerce solution that must support thousands of users or an intranet-based workflow solution limited to internal users, a TP monitor provides a focal point for your application.
- Database applications that must scale to support thousands of users. The NonStop TUXEDO product provides multiplexed access to a single (or multiple) database, resulting in faster overall throughput. In addition, the NonStop TUXEDO product enables access to heterogeneous databases from a single access point.
- Distributed applications that span multiple, possibly heterogeneous platforms (and possibly, heterogeneous databases).
- Traditional on-line transaction processing (OLTP) systems that perform critical business functions for many (often thousands of) users whose activities change the content of many related databases.

The NonStop TUXEDO for Windows NT Server product is a high-performance transaction processing (TP) monitor that enables applications to remain robust and responsive despite peaks in user demand. The NonStop TUXEDO for Windows NT Server product is differentiated from the other TUXEDO products on the market by virtue of its built-in and automatic support for clusters, which means a substantial performance benefit as well as maximum scalability. In addition, the NonStop TUXEDO for Windows NT Server product ensures a more robust application at a lower development cost because the programmer's job is vastly simplified due to the TUXEDO programming and deployment model.

This section provides an overview of transaction processing solutions using NonStop Software for Windows NT Server (hereafter referred to as simply "NonStop Software") products, with particular focus on the NonStop TUXEDO for Windows NT Server software product. The section includes recommendations about alternative design approaches, and information about TUXEDO APIs, as well as information about Jolt and other technologies that support interactive transaction processing — for example, OLTP over the Internet, via a browser. The information contained in this section is high level; for more detailed information, see the *NonStop TUXEDO System Application Programmer's Guide* and related NonStop TUXEDO product documentation.

NonStop TUXEDO for Windows NT Server

The NonStop TUXEDO product is an open, parallel, high-performance enterprise transaction processing monitor that provides the features developers need for distributed online transaction processing applications. Among these features, the NonStop TUXEDO product provides the infrastructure needed to support distributed client/server applications across multiple, heterogeneous hardware platforms and database

management systems using several different inter-application communication styles provided in a single API set.

The NonStop TUXEDO product simplifies distributed application development, deployment, and management, and also supports a two-phase-commit protocol for transactions. A two-phase-commit protocol ensures that the ACID (atomicity, consistency, integrity, durability) properties remain intact despite system failures, which is a key requirement of OLTP systems. (See A Closer Look at OLTP on page B-2 for more information about these properties.)

The NonStop TUXEDO product also brings the key benefit of manageability to distributed, high-volume transaction processing applications. The NonStop TUXEDO product provides a common management environment to ease the burden associated with administering distributed transaction processing applications. Applications comprised of many services, distributed across many servers, can be managed from a graphical user interface, even during runtime.

Tandem's NonStop TUXEDO software supports several application programming interfaces, including the Application-Transaction Manager Interface, the API which is native to the BEA Systems, Inc., TUXEDO product. Because of this compliance, developers who use the NonStop TUXEDO product will be able to easily deploy their application not only on Windows NT Server clusters, but also to some 30 different platforms that support TUXEDO, including UNIX® operating system platforms, Windows NT Server systems, Tandem Himalaya servers, and IBM systems (see Figure 6-2, Three-Tier Heterogeneous TUXEDO Applications Architecture, on page 6-7 for an example figure).

In addition, by incorporating the Jolt for the NonStop TUXEDO product in their solutions, developers can extend the fault-tolerant and manageable NonStop TUXEDO environment directly to Java clients over the Internet or on an intranet, without extra coding.

Note: The NonStop TUXEDO release 1.0 software product is functionally identical to the BEA TUXEDO 6.3 product.The NonStop TUXEDO 2.0 product will expand upon this functionality to include support for clusters of Windows NT Server systems.

This subsection highlights some of the distinguishing features of the NonStop TUXEDO product. For a complete description of the NonStop TUXEDO product, see the NonStop TUXEDO Transaction Services Product Description, available from Tandem's NonStop Software Web site (http://www.nonstopsw.com).

Application Programming Interfaces

As mentioned above, the NonStop TUXEDO transaction processing monitor supports several application programming interfaces. ATMI is the most full-featured and prominent of these API sets, and provides calls for several key distributed processing functions, including:

- Inter-application communications. ATMI provides both request/response and conversational semantics which are used for communications between client and server processes. Request/reply communications can be implemented in an asynchronous or synchronous style.

- Transactions. ATMI provides semantics to mark the beginning and end of activities that should comprise a transaction — a logical unit of work whose activities must all complete or must all not complete — to protect the integrity of the system and its resources.

In addition to ATMI, the NonStop TUXEDO software product supports several interfaces promulgated by X/Open's DTP (distributed transaction processing) model. As discussed in TP Monitor Overview on page 3-11, the X/Open's DTP model defines many of the standard functions provided by TP monitors, including managing transactions; managing resources (such as databases and persistent message queues); and managing communications among applications and between clients and servers. The interfaces defined by X/Open for these functions are TX, XA, and XATMI, respectively. The NonStop TUXEDO product supports these interfaces in addition to ATMI, and in fact, almost all of the X/Open's DTP interfaces and the model itself are based on the BEA TUXEDO product and ATMI. The TX interface provides functions to mark the start and end of a transaction, and to commit, roll back, or abort the transactions. TX is subsumed by ATMI.

Along with the interface libraries, which are provided in both C/C++ and COBOL versions, the NonStop TUXEDO product also includes client and server application templates — the main() program loop for C programs, for example, and template files for COBOL applications. These templates provide predefined subroutines for initializing and terminating servers; allocating buffers to receive and dispatch incoming requests to service subroutine requests; checking the message queue for incoming requests; and dispatching requests to the appropriate service subroutines.

Application developers code service subroutines and application programs and then they build the server using the buildserver utility, which puts the TUXEDO main() program loop together with the service subroutines. The application then is installed on the Windows NT Server system. It requires that the NonStop TUXEDO runtime program be installed and running on the server as well.

NonStop TUXEDO Architecture

The NonStop TUXEDO software product consists of several integrated components and processes as shown in Figure 6-1, including:

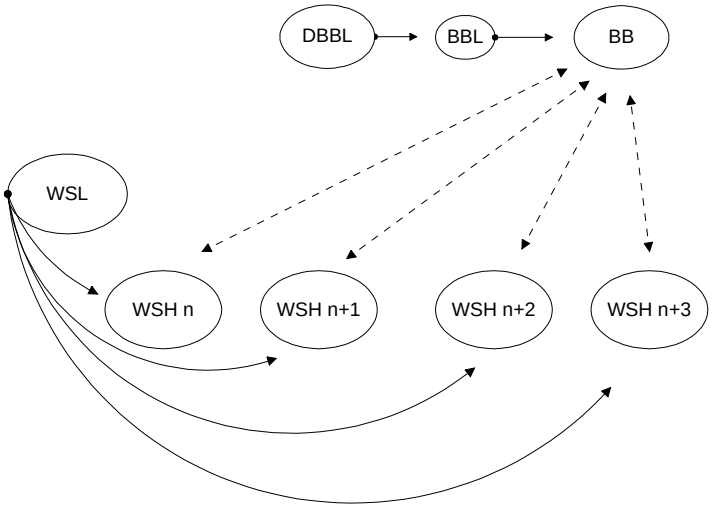
- The **workstation listener** (WSL) process listens (in the same manner as a well-known port listens) for connection requests from remote client processes. When it receives a request, the WSL process assigns a workstation handler to function as a link for communication between the client and the server process. The WSL starts a WSH process if one isn't already available, and can start multiple WSH processes.
- The **workstation handler** (WSH) process acts as the gateway between a workstation client and the server application process, converting ATMI or TX calls made by the client process into calls that will be understood by the NonStop Services Transaction Distributor and NonStop Services Transaction Manager. As shown in Figure 6-1, WSH is a multithreaded process capable of handling multiple clients.
- The **bulletin board** (BB) contains information about the servers and services that comprise the TUXEDO application and about the TUXEDO clients that can connect to the application. The bulletin board comprises a location in memory which is

loaded at boot time and its contents are basically the parameters contained in TUXCONFIG. Both native client processes and server processes attach to the bulletin board. Part of the bulletin board associates service names with the queue address of servers that advertise that service. Clients send their requests to the name of the service they wish to invoke rather than to a specific address. Because there are multiple processors involved in the application, copies of the bulletin board must be maintained on every processor that comprises the entirety of the application.

- The **bulletin board liaison** (BBL) process, which runs on every processor or node of a cluster, keeps its local bulletin board current with information provided by the master bulletin board listener. This **distinguished bulletin board liaison** (DBBL), as it's called, manages the updates to the other bulletin board liaisons.

Information for the bulletin board is obtained from the TUXEDO configuration file for the application. The configuration file contains application identity and resource information; names of servers; and services that comprise the application; and security settings for access to servers and services; among other details that describe the application. System administrators can completely control the application, including dynamically balancing the load, using a graphical management tool. (Dynamic management through the graphical management tool is another one of the key benefits of the NonStop TUXEDO product.)

Figure 6-1. NonStop TUXEDO System/T Components



A copy of the bulletin board (BB) is replicated on every processor that supports the application in a TUXEDO system. The bulletin board keeps track of all the application-wide dependencies —names of the services; where they are located; which users can access them; and so on — as well as status information.

NonStop Services

The TUXEDO architecture provides an enterprise-class framework for developing and deploying distributed OLTP applications, and Tandem adds value to this framework by

delivering the NonStop TUXEDO[1] product on Windows NT Server clusters (n-node[2]) which incorporate the NonStop Services infrastructure layer. The NonStop Services infrastructure layer includes components such as the NonStop Services Transaction Distributor, NonStop Services Transaction Manager, and NonStop Services Distribution Service. The Tandem fundamentals of scalability, high-availability, and manageability derive, in part, from this infrastructure.

For example, the NonStop Services Transaction Distributor starts, stops, and monitors processes; coordinates links between clients and servers; and automatically restarts processes after a failure.

In addition, the Tandem NonStop TUXEDO product provides enhanced manageability. One example of the NonStop TUXEDO product's enhanced manageability is seen in the logging mechanisms. The NonStop Services Transaction Manager implements a single virtual log, an implementation which is unique to the industry. The NonStop TUXEDO product (and also NonStop SQL/MX) makes use of the NonStop Services Transaction Manager and in so doing, creates only a single log file for transactions in which both are involved. Other database implementations involve more than one log file under the same circumstances. With each additional log file, there's more overhead for both normal processing and recovery. Also, because of all the extra overhead, there's the increased likelihood of hung transactions, should a node in the cluster fail.

For more information about the NonStop Services layer, see Tandem's NonStop Services Layer on page 2-9, and see Tandem's NonStop Software Web site (http://www.nonstopsw.com).

Application Architectures

Online transaction processing systems at one time were developed as monolithic applications, but this application design approach has been supplanted by client/server distributed architectures. The three-tier client/server architecture has become the preferred architecture for newly developed and re-engineered enterprise applications for many reasons, including greater scalability of the application. (For additional background information, see Three-Tier Client/Server Model on page 2-4.)

The basic three-tier client/server model is one in which client function (usually, data gathering and display functions only), business logic (at the server process), and the back-end databases comprise three distinct logical entities, or tiers. Note that the tiers do not translate necessarily into three different physical locations, and very often, depending upon design considerations, the second and third tier may reside on the same hardware platform. Solutions developed using the NonStop TUXEDO product, with its clients, servers, and services architecture, are a natural fit for this model.

Three-Tier Client/Server Transaction Processing

In the NonStop TUXEDO client/server model, clients and servers are defined in relationship to each other, and are not defined by location. Client processes have

1. The NonStop TUXEDO software product (version 2.0) will be integrated with Tandem's NonStop Services infrastructure software components.

2. n-node means "greater than 2-node."

limitations on what they can do, depending upon the communications model employed. As mentioned earlier, the two primary models are request/reply and conversational.

Table 6-1 summarizes the functions that can be performed by the client and server in a TUXEDO application environment in terms of these communications styles. In all cases, servers automatically advertise their services when the server is booted.

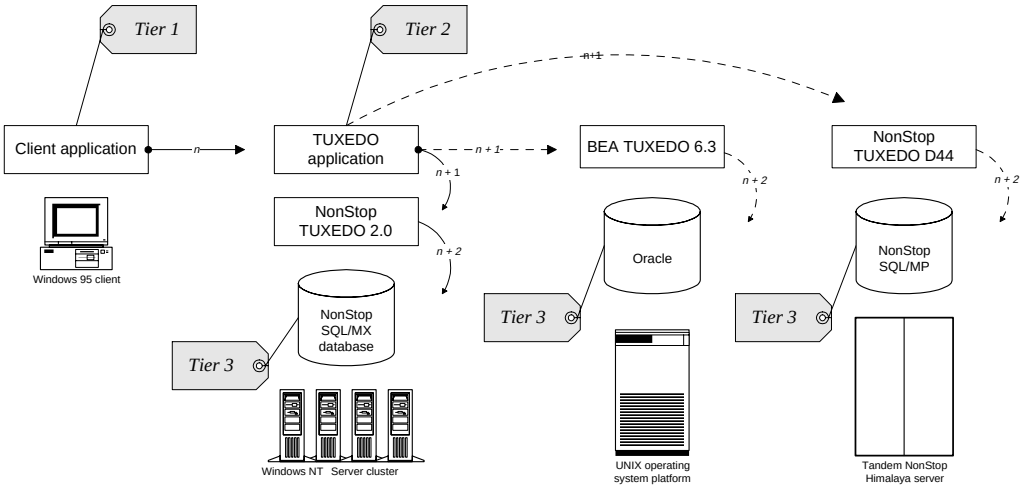
Table 6-1. Functional Summary for TUXEDO Client and Server

Client	Server
Sends request messages	Receives service request messages
Receives reply messages	Sends reply messages (in response to request message)
	Receives reply messages
	Forwards request messages to another request/response server
Initiates a conversation	
	Initiates request/response or conversational requests
	Accepts connection requests (conversational)

Note that a client process cannot:

- Accept connection requests from another conversational client or server
- Receive requests from another request/reply client or server

Figure 6-2. Three-Tier Heterogeneous TUXEDO Applications Architecture



The NonStop TUXEDO product includes application templates for both client and server applications. These templates include the programming logic for many of the basic processing tasks required for distributed client/server applications. The business logic itself is coded as service subroutines into these application templates; the NonStop TUXEDO System/T runtime libraries automatically handle the mechanics of transaction recovery.

NonStop TUXEDO applications are accessed by name; client processes send requests to the name of the service they want to invoke, rather than to a specific address.

Inter-Application Communication

A TUXEDO application running on a single server or cluster is called a TUXEDO administrative domain. A TUXEDO application is defined in a single configuration file; the configuration file describes the machines, servers, and services that comprise the application. (Machine names are logical machine names, and refer to the nodes that comprise a cluster or processors in a multiprocessor.) Essentially, then, the configuration file defines the scope of the application; this scope is also referred to as a TUXEDO administrative “domain.”

To distribute application processing across multiple domains, you must install the NonStop TUXEDO /TDomain gateway. The System /TDomain software product, included with the NonStop TUXEDO product, provides a gateway process that enables distributed transactions between different TUXEDO administrative domains. It is the System /TDomain gateway that enables TUXEDO implementations on disparate platforms to interoperate.

For example, in Figure 6-2, Three-Tier Heterogeneous TUXEDO Applications Architecture, on page 6-7, the three representative platforms would comprise distinct TUXEDO domains. One of the TUXEDO instances is designated as the chief coordinator, or “master,” and the other TUXEDO instances are subordinate. As a general rule for interoperability, the master must be the highest version level of

TUXEDO of all versions (if there are more than one version) running in the group. That is, if a subordinate is running TUXEDO 6.3, the master must likewise be running 6.3 (or higher) as well. Subordinates may be lower versions than the master, but the reverse is not true.

The components of the System /TDomain include:

- Domain gateway server (GWTDOMAIN) for managing communications between a local and remote domain.
- Domain configuration file (BDMCONFIG) for defining services to export, services to import, and security options for each connected remote domain.
- Domain administrative server (DMADM) for managing dynamic changes to the domain configuration.
- Domain administration program (DMADMIN) for making dynamic changes to the domain configuration.

Application developers will note that installing and configuring the System /TDomain gateway software product is an administration task — there is no special coding or application development to be done within the context of the client application or the server applications. (In fact, the /TDomain gateway software is installed automatically when the NonStop TUXEDO product is installed.) Developers will probably work with system administrators to work out the details of the System /TDomain gateway configuration file (BDMCONFIG) and what it should contain in order to link multiple TUXEDO applications (domains) together.

Note: The TUXEDO System /TDomain module is supported in the Windows NT version of the NonStop TUXEDO product at release 1.0 and in the NonStop Himalaya version of the product at NonStop TUXEDO D44.

Interactive Transaction Processing

The Internet, especially the World Wide Web, has created new opportunities for organizations of all sizes to conduct business at any time of the day or night, at the customer or client's convenience. Accessing transaction processing systems from a Web browser client interface is becoming known as "Internet transaction processing" or "interactive transaction processing." Regardless of the name, this is essentially an expanded notion of transaction processing which includes the Internet model, in which thousands of clients may interact with services over a public network and conduct business.

As such, interactive TP has the same requirements as OLTP, but in addition must be able to support virtually any type of client. In addition, because interactive transaction processing via the Internet may expose the organizations' operational processing systems to outside access in this way, there's the need for greater security and manageability.

The NonStop TUXEDO product supports Web access seamlessly, through an add-on product called Jolt. The next subsection describes this in more detail.

Jolt

The Jolt for NonStop TUXEDO product is an interface that extends the TUXEDO application model to the Internet, specifically to Java enabled Web clients. (See Java on page 3-8 for more information about Java.) Composed of several software components, Jolt includes an object-oriented interface to the NonStop TUXEDO product; it is a new API to the TUXEDO product which enables Internet access to transaction processing applications yet exposes none of the application's internal transaction processing programming or semantics.

NonStop Jolt has been shipping since July 1997. For complete details, see the Jolt for NonStop TUXEDO for Windows NT Server product description, available on Tandem's NonStop Software Web site (http://www.nonstopsw.com).

Overview of Application Development

Developing and implementing a TUXEDO software product involves both programming and operations. Developers write the service subroutines into the framework application provided with the TUXEDO product. In addition, however, many variables relating to the application, including how it is distributed across hardware platforms, and which users can use it, are defined in a configuration file.

Servers and service routines offer a structured approach to writing System/T applications. Because many of the communication details are handled by System/T's main(), the application development process is simplified: developers can concentrate on the work performed by the service rather than communications details such as receiving requests and sending replies. In addition, many of the complexities of deploying distributed applications are not handled by the application developer, but in the configuration file at deployment time. Here's an overview of the development process:

1. Design, code, compile server application using C++ or COBOL, choosing the appropriate communications models for the service subroutine and client communications.
2. Use the NonStop TUXEDO buildserver command with ATMI (or TX) header files to create the server executable file. The buildserver command takes the service subroutines and merges them with TUXEDO's main() program loop.
3. Design, code, and compile the client application using C++ or COBOL. The NonStop TUXEDO product includes many sample client applications which can be used as a starting point for your development efforts.
4. Use TUXEDO buildclient command with ATMI header files to create the client executable file.
5. Use a text editor to create (or edit an existing) configuration file. Many deployment issues are handled in this file.
6. Load the configuration file using the NonStop TUXEDO GUI manager.
7. Boot the service application using tmboot or the GUI manager application.
8. Start the client application.

Interprocess Communications

ATMI provides communications infrastructure for several types of client/server communication, including request/response; conversational; and a queued message model. The mode you choose to implement will be based mainly on whether you need to carry context from process to process, in which case, depending upon the specifics of the application, you would likely choose the conversational model.

Here's a brief summary of the primary communications models used by the NonStop TUXEDO product, and how the clients and servers might implement them.

Request/Response Client/Server Communications Model

Request/response services are invoked by service requests along with their associated data. Request/response services can receive exactly one request (upon entering the service routine) and send at most one reply (upon returning from the service routine). In request/response communication, services are started when the server receives a request from a caller, and end when the reply is sent or the request is forwarded.

By definition, clients cannot receive requests nor send replies, but a request/response server can function as a client and send requests to other servers. A client can send any number of requests, and can wait for the replies synchronously or receive replies asynchronously. In certain cases, a client can send a request that has no reply. tpinit() and tpterm() allow a client to join and leave a System/T application.

A request/response server can forward a request to another request/response server. Here, the server passes along the request it received to another server and does not expect a reply. The last server in the chain must send the reply message back to the original requester.

At the end of a particular service routine, a server can send a reply message back to the originating caller, using tpreturn(), or forward the request using tpforward().

Request messages and reply messages are inherently different. The process initiating the request message (the caller, which may be a client or a server process) must provide address information. This context must be maintained by the receiving process and included in the reply message, but the sender of the reply can exert no control over its destination: replies are always returned to the process that originated the request.

The request message can also include a priority setting; if none is defined, a default priority setting, configured for each service, will be used. The server dequeues request messages (from its buffer) according to the priority setting.

The request/response client/server model can be implemented asynchronously, using tpacall(), or synchronously, using tpcall(). In the synchronous implementation, the calling process (typically, the client) waits until it receives a reply message (tpreturn) from the server. In an asynchronous implementation, tpacall sends a request message to a server, but doesn't wait for a reply message; the client continues with other work, perhaps initiating other asynchronous activities.

The asynchronous model must then use tpgetrply to get all reply messages from potentially, multiple servers, and synchronize these replies. By implementing the

tpforward function within the server, request messages can be forwarded from server to server for optimal processing.

Conversational Client/Server Model

Conversational services are invoked by connection requests that include a descriptor which can be used in calling subsequent connection routines. Once the connection has been established and the service routine invoked, either the connecting program or the conversational service can send and receive data as defined by the application until the connection is torn down.

Clients can initiate a conversation but cannot accept a connection request. Servers can receive as well as request connections from conversational client or server processes. Once the connection has been established and the service routine invoked, either the connecting program or the conversational service can send and receive data as defined by the application until the connection is torn down.

The conversation is half-duplex in nature such that one side of the connection has control and can send data until it gives up control to the other side. While the connection is established, the server is “reserved” such that no other process can establish a connection with the server.

As with a request/response server, the conversational server can act as a requester by initiating other requests or connections with other servers. Unlike a request/response server, a conversational server cannot forward a request to another server. Thus, a conversational service performed by a server is started when a request is received and ended when the final reply is sent via tpreturn().

Once the connection is established, the connection descriptor implies any context needed regarding addressing information for the participants. Messages can be sent and received as needed by the application. There is no inherent difference between the request and reply messages and no notion of priority of messages.

A process can initiate both request/response and conversational communication but cannot accept both request/response and conversational service requests.

Queued Message Model

For workflow applications (or other applications in which participants don’t need immediate response or may not be on-line), developers can implement a TUXEDO resource manager, called “System /Q,” which is a queued message facility. This is the basic “store-and-forward” communications model most identified with email systems, for example, but this can be implemented within the context of a transaction processing application as well.

The ATMI function calls that implement this model are tpenqueue, which puts a message in the queue, and tpdequeue, which removes a message from the queue. Both of these functions can be called from any type of System/T application processes: client, server, or conversational. Servers are still designed using either the request/response or conversational model, but in addition, the service subroutine sends messages through the message queue, where they can be held (on disk) until such time as they are moved on

for processing. Messages should be enqueued and dequeued within transactions to ensure one-time-only processing.

The System/Q components themselves must be installed and configured by the system administrator, working in conjuction with application developers. There are two key components that comprise the System /Q server logic: TMQUEUE and TMQFORWARD. Both of these servers are defined in the *Servers section of the TUXCONFIG file for the application.

For complete details about the NonStop TUXEDO product's queued messaging model, see the *NonStop TUXEDO System /Q Guide* or the on-line documentation included with the NonStop TUXEDO product runtime environment.

Transactions and Transaction Processing Functions

A transaction in System/T defines a logical unit of work in which all constituent activities either completely succeed or have no effect at all. A transaction allows work performed in many processes, at possibly different sites, to be treated as an atomic unit of work. It is this transaction "atomicity" that keeps all databases synchronized and consistent, even if application or machine failures occur.

Operations that are to occur within a transaction are delineated by tpbegin() and conclude with either tpabort() or tpcommit(), depending upon the success or failure of all intermediary processing steps.

System/T supports two sets of mutually exclusive verbs for defining and managing transactions: System/T's ATMI transaction demarcation verbs and X/Open's TX interface. Because X/Open used ATMI's transaction demarcation verbs as the base for the TX interface, the syntax and semantics of the TX interface are almost identical to those of ATMI, with a few exceptions. Table 6-2 provides a comparative summary of the transactional features of both ATMI and TX, and the remainder of this subsection describes them in more detail.

Table 6-2. Summary of Transaction Demarcation and Processing Functions (page 1 of 2)

Function Description	ATMI Call	TX Call
Mark the beginning of a transaction.	tpbegin()	tx_begin()
Commit the transaction.	tpcommit()	tx_commit()
Rollback the transaction.	tpabort()	tx_rollback()
Open a resource manager.	tpopen()	tx_open()
Close a resource manager.	tpclose()	tx_close()
Return transaction information; for example, determine whether a transaction is in progress.	tpgetlev()	tx_info()
Suspend the current transaction.	tpsuspend()	~ [can use tpsuspend()]
Resume the current transaction.	tpresume()	~ [can use tpresume()]

Table 6-2. Summary of Transaction Demarcation and Processing Functions (page 2 of 2)

Function Description	ATMI Call	TX Call
Set when tpcommit() or tx_commit() should return.	tpscmt()	tx_set_commit_return()
Set chained or unchained transactions.	~	tx_set_transaction_control()
Set transaction time-out interval.	~	tx_set_transaction_timeout()

ATMI Transactions

ATMI provides three basic transactional primitives:

- tpbegin()
- tpcommit()
- tpabort()

The tpbegin() function identifies the beginning of a global transaction. When a client calls tpbegin(), the NonStop TUXEDO product creates a global transaction identifier, which is associated with subsequent request messages (tpacall or tpcall) from the same client.

As calls are made to services, the transaction identifier and related context is propagated.

When the client completes all the processing tasks successfully, it will invoke tpcommit().

If the client detects a failure (which it will receive as a negative return message from tpcall), it invokes tpabort().

The commit and abort calls notify the NonStop TUXEDO product's transaction manager of the client's request to complete the transaction, either coordinating the commit (of all resource manager updates) or instructing them to roll back.

The NonStop TUXEDO transaction manager processors interact with the NonStop SQL/MX database to complete the coordination protocol.

If the transaction manager detects a failure outside the scope of client initiator, it will force an abort and alert the client.

The initiator can also suspend its work on the current transaction by issuing tpsuspend(). Another process can take over the role of the initiator of a suspended transaction by issuing tpresume(). As a transaction initiator, a process must call one of tpsuspend(), tpcommit(), or tpabort(). Thus, one process can start a transaction that another may finish.

If a process calling a service is in transaction mode, then the called service routine is also placed in transaction mode on behalf of the same transaction. Otherwise, whether the service is invoked in transaction mode or not depends on options specified for the service in the configuration file. A service that is not invoked in transaction mode can define multiple transactions between the time it is invoked and the time it ends. On the

other hand, a service routine invoked in transaction mode can participate in only one transaction, and work on that transaction is completed upon termination of the service routine. Note that a connection cannot be upgraded to transaction mode: if tpbegin() is called while a conversation exists, the conversation remains outside the transaction (that is, as if tpconnect() had been called with the TPNOTRAN flag).

A service routine joining a transaction that was started by another process is called a participant. A transaction can have several participants. A service can be invoked to do work on the same transaction more than once. Only the initiator of a transaction (the process that calls tpbegin() or tpresume()) can call tpcommit() or tpabort(). Participants influence the outcome of a transaction by using tpreturn() or tpforward(). These two calls signify the end of a service routine and indicate that the routine has finished its part of the transaction.

TX Transactions

Transactions defined by the TX interface are nearly identical with those defined by the ATMI verbs. You can use either or both sets of verbs when developing clients and service routines, but you can't use both verb sets within a single process. That is, a process cannot call ATMI's tpbegin() and later call TX's tx_commit()).

The TX interface has two calls for opening and closing resource managers in a portable manner, tx_open() and tx_close(), respectively. Transactions are started with tx_begin() and completed with either tx_commit() or tx_rollback(). tx_info() is used to retrieve transaction information, and there are three calls to set options for transactions: tx_set_commit_return(), tx_set_transaction_control(), and tx_set_transaction_timeout(). The TX interface has no direct equivalents to ATMI's tpsuspend() and tpresume(), but these two ATMI functions can be used in the context of TX transactions.

The TX interface follows the same rules defined for ATMI transactions, but has two additional requirements. Because the default System/T supplied tpsvrinit() calls the ATMI verb tpopen(), if you're going to use TX you must supply your own tpsvrinit() routine to call tx_open(). Likewise, when processing is complete, you must supply your own tpsvrdone(), which calls tx_close().

In addition, the TX interface has two semantics not found in ATMI. These are:

- Chained and unchained transactions
- Transaction characteristics

Each of these is described briefly.

Chained and Unchained Transactions

The TX interface supports chained and unchained modes of transaction execution. By default, clients and service routines execute in the unchained mode; when an active transaction is completed, a new transaction does not begin until tx_begin() is called.

In the chained mode, a new transaction starts implicitly when the current transaction finishes. That is, when tx_commit() or tx_rollback() is called, System/T coordinates the completion of the current transaction and initiates a new transaction before returning

control to the caller. (Certain failure conditions may prevent a new transaction from starting.)

Clients and service routines enable or disable the chained mode by calling tx_set_transaction_control(). Transitions between the chained and unchained mode affect the behavior of the next tx_commit() or tx_rollback() call. The call to tx_set_transaction_control() does not put the caller into or take it out of transaction mode.

Since tx_close() cannot be called when the caller is in transaction mode, a caller executing in chained mode must switch to unchained mode and complete the current transaction before calling tx_close().

Transaction Characteristics

A client or a service routine may call tx_info() to obtain the current values of its transaction characteristics and to determine whether it is executing in transaction mode.

The state of an application process includes several transaction characteristics. The caller specifies these by calling tx_set_*() functions. When a client or a service routine sets the value of a characteristic, that value remains in effect until the caller specifies a different value. When the caller obtains the value of a characteristic via tx_info(), it does not change the value.

Sample Application

There are many sample applications included with the NonStop TUXEDO product. These applications provide a means to validate that the product has been installed correctly, and can be used as a starting point for developing your own applications. Also, as Tandem completes integration of the NonStop TUXEDO and the NonStop SQL/MX products, sample applications will be made available on the NonStop Software Developers' Web Page. (From http://www.nonstopsw.com, select "Developers' Page.")

Recommendations

This section provides some general recommendations about developing applications for the Tandem the NonStop TUXEDO product. See NonStop Software for Windows NT Server Features on page 3-15 for information about features and availability for the NonStop TUXEDO product.

Code Transaction Logic in the Server Process

A transaction can be started by the client process or the server process. Typically the process that starts the transaction should complete the transaction through resolution. However, since the server is essentially the "gatekeeper" to the data, the server should be responsible for data integrity, so the general recommendation is that all transaction processing logic be coded at the server process and not at the client.

Associate Database Requests With Transactions to Protect Database Integrity

You can access resource managers using SQL or ATMI, but only those database requests associated with transactions will be protected by the transaction semantics of the NonStop TUXEDO product. To associate a process with a transaction, your server application must do one of the following:

- Explicitly call tx_begin() to mark the beginning of a transaction
- Implicitly inherit a transaction by receiving a transaction protected message as a result of a tpcall(), tpacall(), tpconnect(), or tpforward() function request from a client

In addition to these programming mechanisms, you can also configure the service (in the bulletin board configuration file) to operate in AUTOTRAN mode, which causes a transaction to start automatically whenever a message is received that is not in transaction mode. AUTOTRAN is not recommended, however, since it adds processing overhead to every message, resulting in degraded performance.

Use TX Rather Than ATMI for Portability

To ensure that the applications you develop will be highly portable to other TUXEDO platforms (UNIX, Tandem NonStop Himalaya, IBM/MVS, and so forth), Tandem recommends that you use the TX API rather than ATMI.

To control multiple transactions from a single application, use the TX API together with the tpsuspend(), tpforward(), and tpresume() functions as described in the next section.

Rules for Using Transaction Functions

Each of the supported APIs in the NonStop TUXEDO product has different semantics for transaction demarcation. Therefore, one general rule is: be cautious in using the different API sets together, and follow these guidelines.

- Do not use transaction demarcation functions from different APIs within the same transaction. If you use the TX API to begin a transaction, you must use the TX API to mark the end of the transaction. For example, use the TX API's tx_begin() and tx_commit(); don't start a transaction with tx_begin() and end the transaction with SQL's COMMIT WORK.
- Use ATMI's tpreturn(TPFAIL) to abort a transaction at the server which was started in the client (ATMI semantics prohibit calling the abort operation (tpabort() or tx_rollback()) in the server for a transaction that was started in the client.)
- The following functions are allowed within transactions demarcated only by TX or ATMI functions: tx_info(), tpgetlev(), tpscmt(), tpsuspend(), tpresume(), and the tx_set_ functions (namely tx_set_commit_return(), tx_set_transaction_control(), and tx_set_transaction_timeout()).
- The tpgetlev() and tx_info() functions determine whether the process is in transaction mode. You can use tpgetlev() regardless of whether the transaction was started by tpbegin() or tx_begin(), or whether the transaction was inherited.

- The tpsuspend() and tpresume() functions are allowed within transactions started by either the tpbegin() or tx_begin() function.
- The tpsuspend() and tpresume() functions cannot be used with an inherited transaction.

Table 6-3. Transaction Demarcation Functions

ATMI	TX	SQL
tpbegin()	tx_begin()	BEGIN WORK
tpcommit()	tx_commit()	COMMIT WORK
tpabort()	tx_rollback()	ROLLBACK WORK
tpopen()	tx_open()	~
tpclose()	tx_close()	~
tpgetlev()	tx_info()	~
tpsuspend()	~	~
tpresume()	~	~
tpscmt()	tx_set_commit_return()	~
~	tx_set_transaction_control()	~
~	tx_set_transaction_timeout()	~

Isolate Code for Unsupported Features

The NonStop Software products are being released in several phases, and the features provided and level of integration will vary, depending upon the versions of the products you are implementing (NonStop SQL/MX 1.0 Beta, NonStop SQL/MX FCS, NonStop TUXEDO 1.0, NonStop TUXEDO 2.0 Beta, NonStop TUXEDO 2.0 FCS).

Be sure to read the release notes and installation notes for any caveats concerning functionality. You can still develop sections of code for unsupported features, but you should comment out the section of code during the compile, and then remove the comment feature when the feature does become implemented.

API Reference

The standard TUXEDO product includes several API choices for developing applications, including the native TUXEDO Application-Transaction Manager Interface (ATMI) and the X/Open's TX interface for transactions. ATMI has been adopted by The Open Group as a standard X/Open API.

ATMI

The Application-Transaction Manager Interface (ATMI) provides the interface between an application and the transaction processing system. ATMI provides routines to open and close resources, such as databases and queue managers; manage transactions; manage typed buffers; and invoke request/response and conversational service calls. All function calls are provided as C/C++ and COBOL libraries with the NonStop TUXEDO product. Table 6-4 summarizes the key ATMI function calls.

Table 6-4. ATMI (Application-Transaction Monitor Interface) List (page 1 of 2)

Type of Operation	C/C++	COBOL Calls	Operation
Application interface	tpinit()	TPINITIALIZE	Join an application
	tpterm()	TPTERM	Leave an application
Buffer management interface	tpalloc()	TPALLOC	Allocate a buffer
	tprealloc()	TPREALLOC	Resize a buffer
	tpfree()	TPFREE	Free a buffer
	tptypes()	TPTYPES	Get buffer type
Request/response communication interface	tpcall()	TPCALL	Send a request, wait for a reply
	tpacall()	TPACALL	Send request asynchronously
	tpgetrply()	TPGETREPLY	Get reply after aynchronous request
	tpcancel()	TPCANCEL	Cancel communications handle for outstanding reply
	tpgprio()	TPGPRIO	Get priority of last request
	tpsprio()	TPSPRIO	Set priority of last request
Conversational interface	tpconnect()	TPCONNECT	Begin a conversation
	tpdiscon()	TPDISCON	End a conversation
	tpsend()	TPSEND	Send data in conversation
	tprecv()	TPRECV	Receive data in conversation

Table 6-4. ATMI (Application-Transaction Monitor Interface) List (page 2 of 2)

Type of Operation	C/C++	COBOL Calls	Operation
Unsolicited notification (error handling) interface	tpnotify()	TPNOTIFY	Notify by client ID
	tpbroadcast()	TPBROADCAST	Notify by name
	tpsetunsol()	TPSETUNSOL	Set unsolicited message handling routine
	tpgetunsol()	TPGETUNSOL	Get unsolicited message
	tpchkunsol()	TPCHKUNSOL	Check for unsolicited messages
Transaction management interface	tpbegin()	TPBEGIN	Begin a transaction
	tpcommit()	TPCOMMIT	Commit the current transaction
	tpabort()	TPABORT	Abort the current transaction
	tpgetlev()	TPGETLEV	Check if in transaction mode
	tpsuspend()	TPSUSPEND	Suspend the current transaction
	tpresume()	TPRESUME	Resume the current transaction
	tpscmt()	TPSCMT	Set when tpcommit() or tx_commit() should return
Service routine template	tpservice()	TPSVCSTART	Start a service
	tpreturn()	TPRETURN	End service routine
	tpforward()	TPFORWAR	Forward request and end service routine
Dynamic advertisement interface	tpadvertise()	TPADVERTISE	Advertise a service name
	tpunadvertise()	TPUNADVERTISE	Unadvertise a service name
Resource manager (database) interface	tpopen()	TPOPEN	Open a resource manager
	tpclose()	TPCLOSE	Close a resource manager

TX

The X/Open distributed transaction processing (DTP) model specifies the TX interface for transaction demarcation. The ATMI interface subsumes the TX interface, so you don’t have to do anything different to implement these calls than you would using ATMI; you will still include the atmi.h header file in your C program, for example.

Table 6-5. Summary of X/Open TX API

Function Description	Function call
Marks the beginning of a transaction.	tx_begin()
Commits the transaction.	tx_commit()
Rolls back the transaction.	tx_rollback()
Opens a resource manager.	tx_open()
Closes a resource manager.	tx_close()
Returns transaction information; for example, determine whether a transaction is in progress.	tx_info()
Sets when tpcommit() or tx_commit() should return.	tx_set_commit_return()
Sets chained or unchained transactions.	tx_set_transaction_control()
Sets transaction timeout interval.	tx_set_transaction_timeout()

SQL

The NonStop TUXEDO product also supports SQL transaction calls within the context of client and server applications.

Table 6-6. Summary of SQL Transaction Demarcation Functions

Function description	Function call
Marks the beginning of a transaction	BEGIN WORK
Commits the transaction	COMMIT WORK
Rolls back the transaction	ROLLBACK WORK

Client/Server Development Tools Reference

Many vendors specialize in providing the development tools needed for modeling, designing, developing, or generating three-tier (or multi-tier) distributed client/server systems, including transaction processing systems. Some of these tools are listed in Table 6-7; the tools listed include a range of products, from development environments to full-featured integrated CASE (computer aided systems engineering) tools.

For example, DYNASTY is an object-oriented cross-platform client/server development environment that supports multiple operating systems, databases, and TP monitors, including TUXEDO; this product can be used to generate the code required to interface to the ATMI. This information is being provided for developers to use as a starting point for their own research, and being listed in the table does not confer an endorsement by Tandem for any product.

Table 6-7. Development Tools for Distributed Applications

Vendor	Example Product Name	Web Address
Bay Technologies Pty. Ltd	TPCharger	http://www.baytech.com.au
BEA Systems, Inc.	BEA Builder, Jolt	http://www.beasys.com
BMC Software	Patrol® Management Suite	http://www.focalpoint.com
Borland	Delphi, C++ Builder, JBuilder	http://www.borland.com
Compuware Corporation	UNIFACE	http://www.compuware.com
Dynasty	DYNASTY	http://www.dynasty.com
Fujitsu ICL Trading	TUXEDO Visual Basic Toolkit	http://www.fujitzu-icl.com
IBM	VisualAge	http://www.ibm.com
Magna Software Corp	MAGNA X	http://www.magna.com
Micro Focus	Micro Focus COBOL	http//www.mfltd.co.uk
Microsoft	Visual Basic	http://www.microsoft.com
Neuron Data	Elements Enterprise/C	http://www.neurondata.com
Oracle	Developer/2000, Designer/2000	http://www.oracle.com
ParcPlace	VisualWorks	http://www.parcplace.com
Planetworks	Interspace	http://www.planetw.com
Prolifics (A JYACC Company)	JAM, JAM/WEB	http://www.prolifics.com
Seer	Seer*HPS®	http://www.seer.com
Sterling Software	COOL:Gen (formerly known as Composer), KEY:ObjectView	http://www.sterling.com
Sybase/Powersoft	PowerBuilder	http://www.sybase.com
Tandem	Tandem ADE for Borland C++	http://www.tandem.com
Unify	Vision	http://www.unify.com

Table 6-8. Testing Tools for TUXEDO Applications

Vendor	Example Product Name	Web Address
Mercury Interactive Corporation	LoadRunner, WinRunner, XRunner	http://www.merc-int.com
Rational (formerly Pure Software, Pure Atria)	EMPOWER, Visual Quantify, PurePerformix, Performix/Web	http://www.rational.com

Table 6-9. Migration, Interoperability, and Mainframe Connectivity Tools

Vendor	Example Product Name	Web Address
Bay Technologies Pty Ltd.	OPEN GATEWAY	http://www.baytech.com.au
Fujitsu ICL Trading	TP Manager	http://www.fujitsu-icl.com
Tandem	TUXEDO to Pathway Gateway	http://www.tandem.com

Table 6-10. Management Tools for TUXEDO Applications

Vendor	Example Product Name	Web Address
BMC Software	Patrol, PATROL Knowledge Module for TUXEDO	http://www.focalpoint.com
Oki Electric (Japan)	DressUP	
	InfoWatch/TP™ for TUXEDO	http://www.focalpoint.com
	ISAM-XA	

A Windows NT® Server Cluster Notes

This section provides background on technologies relevant to the NonStop Software for Windows NT Server product line, including various processor architectures; clusters on Windows NT Server as provided by Microsoft Cluster Server (MSCS) and the MSCS[1] API; and technologies that provide the physical connectivity between nodes in a Windows NT Server cluster: for example, the Tandem ServerNet™ interconnect mechanism. The section starts with a brief look at some of the factors that have led to the great interest in Windows NT Server as an applications server platform and at why NonStop Software for Windows NT Server products are the best choice for building enterprise-class applications on the Windows NT Server platform.

Why Windows NT Server?

As a provider of hardware and operating systems, Tandem has been most successful in enterprise computing environments that demand no less than 24-hour-a-day, 365-day-a-year operations — hence the word “NonStop” in Tandem product names. The result of this success is that, worldwide, Tandem hardware and software solutions handle 66% of all credit card transactions, 80% of all ATM (automated teller machine) transactions, and 90% of securities transactions. Such environments are truly “mission critical” and can afford no outages. Other applications that must be highly available include point-of-sale (POS) applications, customer support call centers, and emergency (911) services.

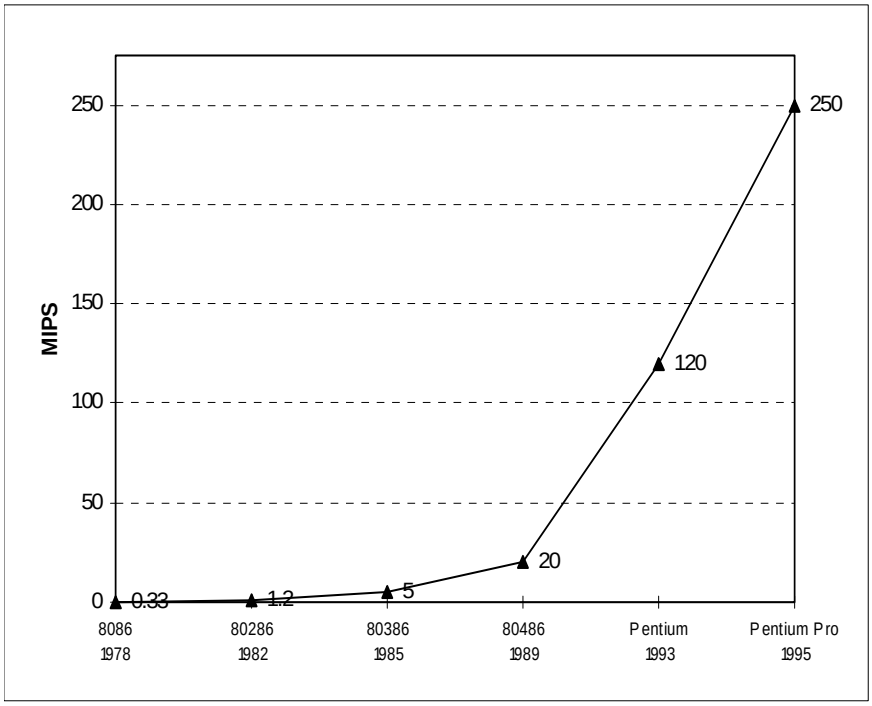
While Tandem has been building hardware and software to support mission-critical global financial infrastructure, some other key vendors have been working toward different goals. One significant development has been the advancement of the Intel microprocessor. Built around an open computing architecture, Intel microprocessors have become successively more powerful and affordable, which has led to low-cost, high-performance hardware and helped fuel the demand for more advanced software to harness that power.

Intel Processor Price/Performance

Intel processors provide exceptional value for desktop computers, workgroup servers, and departmental applications servers. Each successive generation of Intel processor is over twice as fast as its predecessor. Capable of processing over 250 million instructions per second, today’s Intel Pentium Pro processor is 750 times faster than its ancestor, the Intel 8086, and over twice as fast as the Pentium processor inside a desktop workstation.

1. Formerly known by its code-name, the “Wolfpack” APIs.

Figure A-1. Intel Processor Improvements



The Intel Pentium Pro microprocessor has a 4-way bus architecture, enabling multiple Pentium Pros — typically between two and sixteen — to be contained in a single hardware enclosure. The bottom line is that Intel-based hardware is a powerful, cost-effective workgroup or small enterprise applications server. (For more information about Intel processing, see Intel's Web site: http://www.intel.com).

Another significant development has been Microsoft's development of DOS and then Windows around the Intel processor. Whether it's Windows 3.11, Windows 95, or Windows NT Workstation, Microsoft Windows in one form or another has become the operating system of choice for the enterprise Intel-based desktop computer. In 1993, Microsoft first released Windows NT Server operating system. Built on a microkernel architecture, Windows NT Server was designed for distributed environments from the start (see Windows NT Server Architecture on page A-9 for more information). The Windows NT Server operating system has become more robust with each successive version and is emerging as a cost-effective business application server platform.

Windows NT Server Is Maturing

The Windows NT Server network operating system has evolved from a basic file, print, and email server to an applications server platform, and is beginning to compete with UNIX system servers for a growing share of the applications and database server market. Windows NT Server and Microsoft's companion BackOffice products, comprising products such as Exchange, SQL Server, SNA Server, and Systems Management Server, have enabled Microsoft to compete aggressively in the network

operating systems market against products such as Banyan VINES, Novell NetWare, and IBM LANServer, and to compete against the applications server market dominated by UNIX system servers.

Arguments comparing the Windows NT Server operating system to UNIX system abound, with proponents on either side highlighting the benefits and downplaying the respective weaknesses. However, the facts remain: information technology decision-makers are not only evaluating Windows NT Server, they are also deploying it in large numbers, and they plan to continue to do so and use it not only for file, print, and email, but also as an applications server. IDC (International Data Corporation, Framingham, Massachusetts) reports sales of Windows NT Server more than doubled in 1996, up from 363,000 in 1995 to 725,000 in 1996. And Dataquest has forecast that by 1999, Windows NT will have 41% of the applications server market compared to 2% for UNIX ("UNIX vs. Windows NT" *Byte* magazine, May 1996).

Windows NT Server is an attractive applications platform for many reasons, starting with its relatively low cost. At street prices of less than $500 per server and well under $50 per client license (May 1997), Windows NT Server includes a fully integrated Web and ftp server (Internet Information Server, or IIS), file, print, and other basic services and utilities. The low entry price for the software, coupled with the fact that it runs on low-cost hardware from dozens of vendors, makes it virtually "open" (in the non-proprietary sense of the word). For example, Windows NT Server currently runs on Intel, Digital Alpha, PowerPC, and MIPS processors (although support for these last two is being abandoned by Microsoft in upcoming releases).

Windows NT Server is also cost-effective in terms of configuration and management; installation is simple, configuration is almost plug-and-play, and administration is relatively easy, depending upon the particular network configuration. Furthermore, Windows NT Server enables organizations to leverage the expertise of their Windows application developers (a sizable group) using Visual Basic, PowerBuilder, Visual C++, and, more recently, Visual J++.

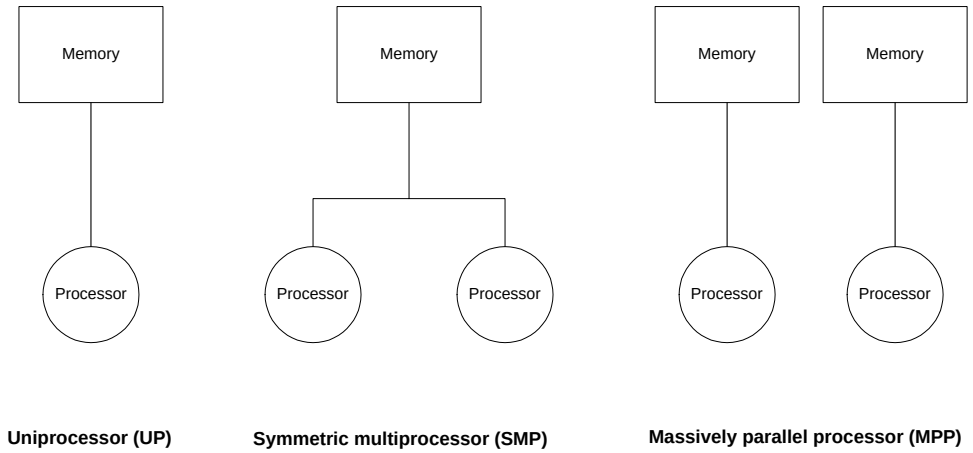
However, as a symmetric multiprocessing operating system (SMP), Windows NT Server won't scale to meet the needs of mission-critical computing as described in the next section.

Processing Architectures Affect Scalability and Availability

From a high-level viewpoint, there are basically three different types of business computing architectures, each with its own advantages and disadvantages. From simplest to most complex, these include uniprocessor and two distinct types of multiprocessor architectures: symmetric multiprocessor (SMP) and massively parallel processor (MPP). As its name implies, the uniprocessor architecture relies on a single CPU, and performance ceilings in uniprocessor-based hardware depend on the processing capabilities of the single CPU in the hardware enclosure.

On the other hand, multiprocessor architectures allow multiple CPUs to be combined or added incrementally so that, as the processing demands increase, the hardware can scale to meet the additional demands of the workload. However, there are two different types of memory model that can be implemented in multiprocessor architecture, as shown in Figure A-2.

Figure A-2. Processor Architectures



Uniprocessor and SMP systems share memory; MPP systems do not.

SMP architectures rely on a shared-memory model; each additional processor uses the same global memory shared by the other processors. This results in fast communication between processes, but it also means that as applications and the operating system take advantage of the increasing number of processors in an SMP system as the system grows, the local cache is repeatedly emptied and filled as tasks are scheduled and interrupted for execution on a given processor. Due to this cache coherency requirement, throughput can actually decrease rather than increase after a certain number of processors have been added to the system. Thus, SMPs have limited, non-linear scalability.

On the other hand, an MPP architecture doesn't share memory; for interprocess communications to occur, the operating system sends messages between the components of a processor-and-memory combination over a high-speed interconnect mechanism. Because neither the local memory nor the processors are shared, MPP systems that use this approach have far greater scalability than SMP systems.

Tandem's NonStop Kernel, the operating system for the Tandem Himalaya server, is an MPP operating system designed from the ground up for high availability and scalability. Up to 16 processors can be installed in a Tandem Himalaya server, and a network of these can be tied together for additional scalability. Most significantly, because of the "shared-nothing" architecture, there's an almost 1:1 performance improvement ratio for each additional processor.

Clustering Technology Provides Availability

System architecture affects not only scalability, but availability as well. High availability means that a server system can minimize the impact of any failure and recover from it automatically and transparently to system users. Intel processor-based server hardware implementations, with their "hot-swappable"[1] RAID 5

1. Capable of being removed and replaced while the system remains operational.

implementations, UPS (uninterruptable power supply) systems, and other fault-tolerant mechanisms, have come a long way toward achieving greater availability in recent years.

In addition to hardware mechanisms, software features built into an operating system or in the software applications can ensure greater availability by design. Depending upon the design, both planned and unplanned outages (application upgrade, migration, and administration tasks) can have minimal or no negative impact on operations. For example, a relational database management system, such as Tandem NonStop SQL/MP, that supports online administration will offer greater availability than one that does not.

Tandem Process Pair Mechanism Provides Greater Availability

Availability doesn't depend on processor architecture alone; the design of the operating system also affects availability. The NonStop Kernel operating system provides high availability through the use of process pairs, an architectural construct designed and patented by Tandem. A process pair consists of a primary and a backup process running in separate processors. The primary process sends checkpoints — messages containing state information — to the backup process. If the primary process fails, regardless of cause, the backup process takes over the functions of the primary process.

This approach to process takeover is referred to as "fail fast," which means that the backup process anticipates the failure and takes over without affecting the end-user interaction or losing data. Fail fast also prevents data corruption by ensuring that errors are not propagated through the system.

Process pairs are used extensively through the NonStop Kernel software[1]. Process pairs are embedded in many layers of Tandem NonStop system infrastructure, which means that in many cases, applications gain the benefits of availability much more easily. For example, in the Tandem NonStop Kernel, the disk access manager provides reliable access to data despite processor, channel, controller, or disk failures. The Tandem NonStop Kernel has been designed to run on Tandem NonStop Himalaya servers, a clustered MPP implementation; each hardware enclosure contains from two to sixteen processors connected by high-speed bus.

Microsoft Cluster Server and Windows NT Server Clusters

Clustering is the approach Microsoft is taking to extend and enhance the scalability and availability of the Windows NT Server operating system. Although Windows NT Server is a robust network operating system, the underlying SMP architecture doesn't provide the level of performance that enterprise-class applications demand. By joining multiple Windows NT Server systems together over a high-speed interconnect mechanism into clusters, application designers can make Windows NT Server systems highly available and scalable.

To enable multiple Windows NT Server systems to be combined into loosely coupled clusters of SMP nodes, Microsoft has developed a clustering services API, MSCS

1. This is greatly oversimplified for purposes of this discussion. The Tandem NonStop Kernel actually supports three levels of process availability, described as *immediate persistence*, in which one process immediately notifies a backup process to take over in the event of a failover; *initialized persistence*, in which the backup process is already initialized and ready to take over; and *process pairs with continual checkpointing*, in which context information is routinely sent from the primary process to the backup process. Immediate persistence takes longer to recover from than initialized persistence, while continual checkpointing provides fastest recovery.

(Microsoft Cluster Server, formerly code-named Wolfpack). An MSCS cluster is a logical server built from multiple, separate computers. An MSCS cluster node is a Windows NT Server that is a constituent of the cluster; the node can be uniprocessor-based or SMP processor-based. Tandem and several other vendors are partnering with Microsoft to develop MSCS and ensure that MSCS delivers the high availability and scalability required to support critical business applications.

MSCS will make use of local-area network technology, such as Ethernet network interface cards and cable, for physical communication between the two Windows NT Server boxes. Other network interfaces will be supported as well, including FDDI (Fiber Distributed Data Interface) and the Tandem ServerNet interconnect mechanism. (ServerNet is an interface card that can be installed in peripheral component interconnect (PCI) bus hardware to provide a much higher speed interconnect mechanism between nodes in a Windows NT Server cluster than competing approaches, such as Ethernet or FDDI.)

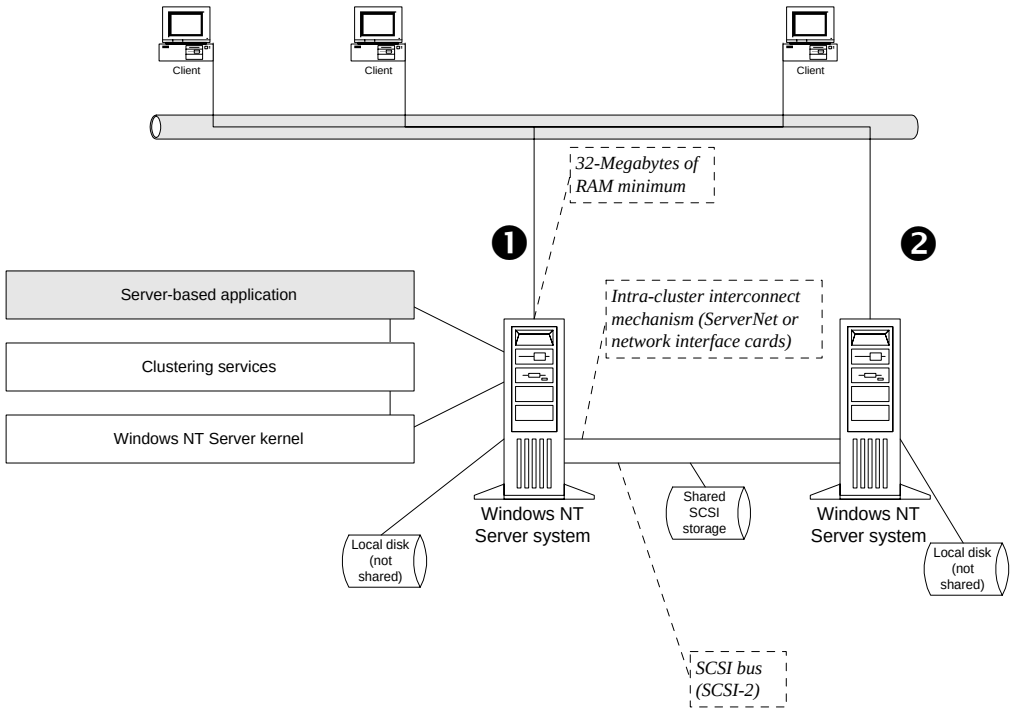
For best overall throughput, Tandem recommends using the Tandem ServerNet interconnect mechanism. Compaq, Dell, Oracle, Unisys, and many other hardware and software vendors are licensing or have endorsed ServerNet as a robust interconnect mechanism for their Windows NT Server environments. Tandem is also working to have ServerNet adopted as an industry standard. And although ServerNet is not required, developers who may be thinking about deploying an application on both Windows NT Server and Tandem NonStop Himalaya systems should note that the ServerNet PCI card is the same technology that is used in Tandem Himalaya server hardware, and that the long-range goal is to achieve interoperability between Tandem NonStop Himalaya servers and Windows NT Server clusters over the ServerNet bus. (For more information about ServerNet, see Tandem's Web site, at www.tandem.com.)

Microsoft plans to release MSCS in two phases. Phase 1 provides fail over between two Windows NT Server nodes only. An application, file service, print service, or other resource can be configured to run on one node and restart on the second node in the event of a failure.

How Fail Over Works

Fail over in Windows NT Server systems requires that one node of the cluster be identified as the primary node and the other node be identified as the backup node. Each node must be aware of the other node's condition on a continuous basis; this "awareness" is accomplished by means of a "heartbeat" mechanism. The heartbeat mechanism consists of "I'm alive" RPC messages (conceptually similar to "keep alive" and "ping" packets used by low-level network protocols) sent at regular intervals from the primary to the backup node in the cluster. If one or more messages is missed during this interval, the cluster initiates the *regroup algorithm*, an error-recovery algorithm designed and patented by Tandem that ensures that the backup node correctly detects, identifies, and distinguishes the failed and operational nodes. (Note that in MSCS phase 1 there can only be two nodes; one primary, one backup. If both fail, there is no further recovery.)

Figure A-3. MSCS APIs Enable Clustering of Windows NT Server Systems



MSCS phase 1 provides fail over between two Windows NT Server systems. File shares, print services, mail services, other applications — such as the NonStop Software-based solutions you develop — and IP addresses can fail over from node 1 to node 2 of the cluster in the event of a failure.

Microsoft has extended the heartbeat mechanism to allow applications to restart if they fail. The MSCS resource monitor periodically polls DLLs (dynamic link libraries) to check for application failures. If a DLL determines that the application is not responding, MSCS can restart the application. Initially, MSCS will monitor Windows NT Server file shares and Microsoft's Internet Information Server.

Fail Over Compared to Fail Fast

Although fail over and restart provides many benefits — for example, the application will restart quickly, typically without the need for operator intervention — node fail over does not protect the application from the effects of a failure. Rather, the application must be restarted and reinitialized in the new node. Open files may be corrupted, and session information is lost. The application must recover the lost information. Clients must reconnect, re-establish their sessions, and redo any work that had not been saved.

Such response may be acceptable for many workgroup applications, such as email, file services, and print services, but for mission-critical applications — particularly in the financial services and emergency services sectors — fail over is not sufficient. Banks, stock exchanges, and 911 emergency response services cannot wait several minutes for node fail over, application restart, and database recovery.

In developing the NonStop Software for Windows NT Server product set, Tandem has also ported key portions of its message and file handling system to the Windows NT Server platform. This “NonStop Services” layer provides process creation and checkpointing facilities and enables NonStop Software products to run as process pairs on Windows NT Server clusters. If a node fails, NonStop Software receives the “node down” message to enable the backup disk and transaction management processes to recover transparently to the application. Furthermore, the NonStop Services layer brings n-node clustering to Windows NT Server today.

Two-Node Compared to *n*-Node

Fail-over support will enable Windows NT Server to provide higher availability than it currently does without it, but fail over does not provide for scalability. MSCS phase 2, expected to be released by Microsoft during 1998, will enable clustering of up to 16 Windows NT Server systems. Thus, when an application exceeds the performance capacity of a single Windows NT Server system, one could add a second SMP box, a third SMP, and so on, thereby achieving linear scalability. MSCS phase 2 will also enable load balancing and parallel processing required for scalable distributed applications. For more information about MSCS, see Microsoft’s Web site at www.microsoft.com.

Tandem’s NonStop Himalaya server systems have a proven track record of providing continuous operations, high availability, and scalability. For example, Tandem’s process pair mechanism provides software fault tolerance automatically for many subsystems in the NonStop Kernel operating system. The issues are discussed in detail in “NonStop Availability for NonStop Himalaya and Windows NT Server Systems,” available on Tandem’s Web site at http://www.tandem.com.

Table A-1 summarizes some key operating system features and the respective names on each platform.

Table A-1. NonStop Kernel and Windows NT Server System Highlights

	NonStop Kernel	Windows NT
Basic architecture	Message passing	Shared memory
Protected execution space	Yes (Privileged)	Yes (Kernel Mode)
Processes outside the execution space	Yes (Non-priv)	Yes (User Mode)
APIs exposed to programmers	Environments (Guardian; OSS*)	Environmental subsystems (Win32, OS/2, POSIX)
Hardware abstraction layer	no	yes
Fault tolerant mechanism (software)	Process pairs	Microsoft Cluster Server (MSCS)**
Type of software fault tolerance	Fail fast	Fail over***

* OSS is Tandem's version of POSIX.

** Tandem NonStop Software for Windows NT Server does not require Microsoft Cluster Server.

*** Fail fast with NonStop Software for Windows NT Server.

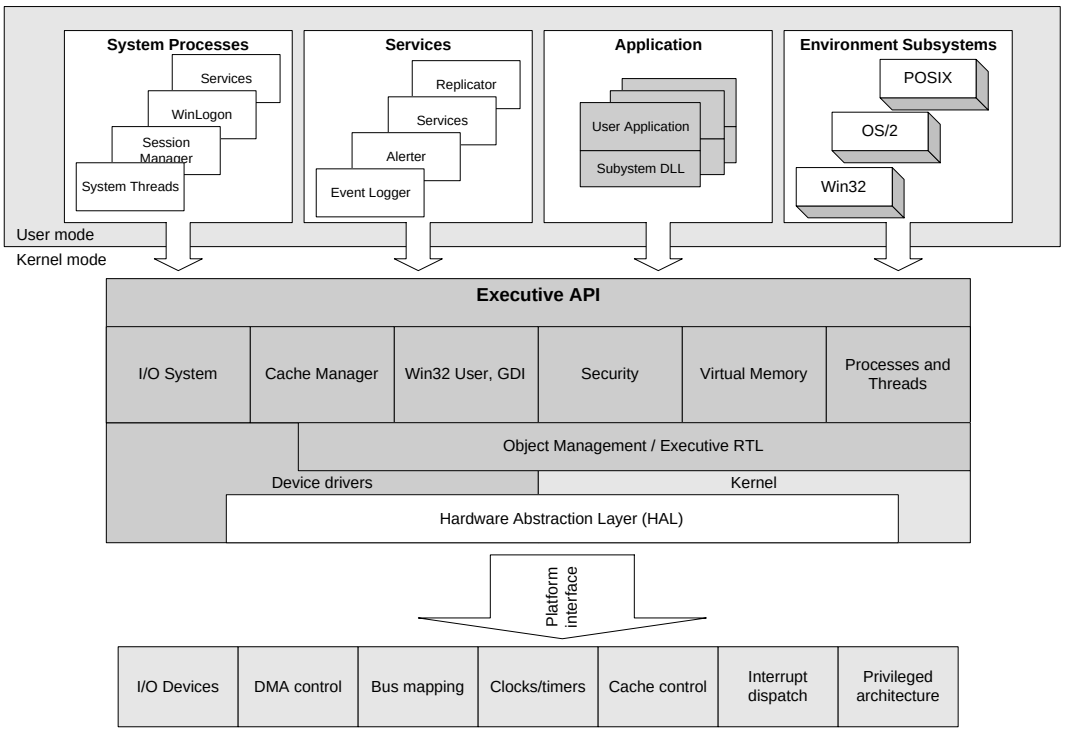
Windows NT Server Architecture

NonStop Software products run on the Windows NT Server operating system. The Microsoft Windows NT Server operating system is a 32-bit multithreaded, multiprocessing operating system. The operating system architecture is a layered, modular operating system that supports both uniprocessor and symmetric multiprocessor (SMP) application architectures. With additional software, including Tandem's NonStop Software for Windows NT Server or Microsoft Cluster Server (MSCS), announced for release in Windows NT Server/E (Enterprise Edition), multiple Windows NT Server systems can be combined into clusters.

Windows NT Server Architecture

The Microsoft Windows NT Server network operating system currently runs on a variety of processor models and architecture types, including Intel-family uniprocessor and SMP, as well as PowerPC, Digital Alpha, and MIPS. (In late fall 1996, Microsoft announced it will be abandoning the PowerPC and MIPS processors, in favor of further development in the Intel and Alpha lines.) The Windows NT Server operating system provides high-performance file, print, database, messaging, and application services.

Figure A-4. Windows NT Server Architecture



The Windows NT operating system is a modular operating system that implements the client/server design model: that is, the operating system is divided into several processes, each of which provides a different set of services to layers that request them (see Figure A-4).

According to Microsoft[1], "one of the design goals of the Windows NT operating system was to keep the base operating system as small and efficient as possible." The operating system is organized into two broad divisions: the Kernel mode, which runs as protected process space; and the User mode, which is the nonprivileged processor mode. The Windows NT kernel manages basic operating system functions including scheduling and dispatching threads, synchronizing multiprocessors, and handling hardware exceptions.

The Kernel contains just the essential base operating system functionality. The kernel has access to system data and hardware. The Windows NT Executive, also part of the kernel, provides a message-passing facility that enables servers to communicate with application processes. The Executive comprises a series of components that implement virtual memory management, object (resource) management, I/O and file systems (including network drivers). Each of these components calls the others through a set of carefully specified routines.

1. Microsoft Windows NT Server Resource Kit. *Windows NT Server Networking Guide*. Microsoft Press. 1997.

The nonprivileged user mode is where applications such as NonStop SQL/MX and NonStop TUXEDO for Windows NT Server run. The subsystems contained in this process area provide APIs for the Win32, OS/2, POSIX, and security subsystems.

These are just some of the high-level details of the Windows NT Server architecture. For complete information about the Windows NT Server operating system, refer to Microsoft publications such as *Advanced Windows: Third Edition* and the Windows NT Server Resource Kit.

B Enterprise Computing Primer

"Mission-critical" or "enterprise-class" distributed applications usually means one of two key types of applications — decision support systems (DSS) and online transaction processing (OLTP) systems. Online transaction processing systems typically work with operational data; financial operations, order entry, scheduling, point-of-sale systems, and bank card withdrawals and deposits are examples of systems that manage operational data.

On the other hand, decision support systems, which include a range of applications such as data warehouses, executive information systems, and online analytical processing environments, typically work with data that has already been processed in some manner. Such data is called *informational data*, or simply information, to distinguish it from the *operational data* used in the online transaction processing systems.

Decision support systems are typically created from data that has been extracted from operational systems. Users query the DSS application looking for buying habits, trends, and so forth. If the information isn't up-to-the-minute current, chances are the results won't be cause for problems. Nonetheless, decision support and data warehouse systems are strategic systems that enable companies to leverage information to make better business decisions. A data warehouse with sufficient detailed data to make important decisions is typically 50 gigabytes or larger.

The basic features of DSS compared to OLTP applications are shown in Table B-1.

Table B-1. Typical DSS and OLTP Application Characteristics

	DSS	OLTP
Data is updated frequently	No	Yes
Predefined structured access to data	No	Yes
Large data sets	Yes	Yes
Large number of concurrent users	No	Yes
Large number of relatively short interactions	No	Yes
Large-size data types	Yes	No
Ad-hoc data queries	Yes	No
Recent, aggregated, derived, or historical data	Yes	No
Transactions distributed across multiple platforms	No	Yes
Typical users include analysts, managers, decision makers	Yes	No
Typical users include data entry clerks, other systems, or consumers (using the Internet)	No	Yes

A Closer Look at OLTP

Online transaction processing systems, as the name suggests, are *transaction*-oriented: A transaction comprises a *logical unit of work* in which a business activity causes multiple changes to operational data. Key to transactions and the logical-unit-of-work concept is the requirement that transactions maintain the "ACID" properties:

- Atomicity: The entire transaction must be completed or aborted.
- Consistency: The transaction takes a computing system from one consistent state to another.
- Isolation: A transaction's effect is not visible to other transactions until the transaction is commmitted.
- Durability: Changes made by a committed transaction are permanent and should tolerate system failures.

A two-phase-commit (2PC, or 2pc) protocol ensures that operational data retains the essential ACID properties. Typically implemented in transaction management software, a two-phase-commit protocol is essentially a mechanism through which processes coordinate distributed transactions, through two distinct phases. One of the processes in a transaction involving multiple participants must coordinate all the other processes. In the first phase, or *prepare phase*, the coordinating process tells all the other processes to prepare to commit. The coordinator must receive acknowledgements from all participants in the transactions for the prepare phase to go to the *commit phase*; if not, the transaction will be aborted.

This discussion greatly oversimplifies many OLTP concepts, but nonetheless, there's nothing particularly unique to information technology about the ACID concept or the notion of the logical unit of work; these concepts are basic accounting principles at work. For example, if an asset account is debited, the appropriate liability account (and sometimes also an equity account) must be credited or the books will be out of balance. It is not acceptable for one part of the process to succeed and the other to fail. Another example can be seen in contract law: each party to a contract must agree to the terms of the contract, or none of the parties is bound by the terms of the contract.

One assumption in this discussion is that the transaction is *distributed* across multiple systems. Maintaining transaction integrity in a single platform, stand-alone batch system is a lot easier than on a distributed system that spans multiple server platforms and is accessed by hundreds or thousands of users in real time over a network. The chances for a variety of failures and problems escalate with the complexity of the networked distributed system.

Despite the different requirements for OLTP and DSS data, the boundaries are beginning to blur in a couple of key areas. For instance, analysis is beginning to take place on operational data as well as on data extracts. And thanks to the Internet, thousands of users (rather than just handfuls of business analysts) may potentially attempt to access a decision support system or data warehouse hosted via Web server interface.

Furthermore, both types of applications must be able to serve large numbers of clients, handle large quantities of data, and recover from problems in sub-second timeframes. In other words, both types of applications must be *highly available* and *scalable*.

Benefits of a TP Monitor

In general, transaction processing (TP) monitors bring the benefits of resource management to the distributed computing environment. Applications comprised of many services, distributed across many servers, can be managed from a graphical user interface, even during runtime.

In addition, a TP monitor such as the NonStop TUXEDO for Windows NT Server software product provides a consistent application-to-application interface in a distributed cooperative processing environment. NonStop TUXEDO (and all other vendors' TUXEDO products) insulates application developers from low-level synchronous and asynchronous primitives, such as sockets programming. NonStop TUXEDO for Windows NT Server programmers use the high-level ATMI API, a set of about 30 calls, to develop applications that can span servers, services, and other applications.

A TP monitor that supports a two-phase-commit protocol ensures the ACID properties of all transactions, even across different database management systems; transaction IDs and context are sent between applications.

TP monitors not only handle distributed transaction processing, but inter-application messaging and other enterprise-wide application services as well. TP monitor software enables applications to encompass heterogeneous databases as well as non-DBMS data.

According to the Gartner Group, TP monitors built on open systems are evolving into general-purpose, production-oriented subsystems for distributed computing[1]. For example, Microsoft Transaction Server (MTS, formerly code-named "Viper") is built on an underlying object-request broker technology: specifically, the DCOM (Distributed Component Object Model).

Both BEA TUXEDO and Tandem NonStop TUXEDO for Windows NT Server (and other platforms) are evolving to provide object broker mechanisms. This evolving model is referred to as *object transaction manager.* Thus, transaction-oriented applications (transaction-oriented in the "logical unit of work" sense) aren't the only applications that can benefit from a TP monitor.

1. "Middleware: The Foundation for Distributed Computing." Gartner Group Client/Server Strategic Analysis Report. Natis, Schulte, Light. October 25, 1996.

Glossary

ActiveX	A marketing name for a broad range of Microsoft technologies that rely on Microsoft's COM/DCOM technology. Many Microsoft technologies that were formerly thought of as OLE technologies have been re-labeled with the ActiveX brand.
ADO	ActiveX Data Objects.
ANSI	American National Standards Institute.
API	Application Programming Interface.
ATL	Active Template Library.
ATMI	Application-Transaction Manager Interface.
COM/DCOM	Component Object Model/Distributed COM.
CORBA	Common Object Request Broker Architecture.
DAO	Data Access Objects.
DCE	Distributed Computing Environment.
DSS	Decision Support Systems.
DTC	Distributed Transaction Coordinator. Microsoft's implementation of a two-phase commit protocol.
IIS	Internet Information Server. Microsoft's implementation of the http (hypertext transfer protocol) service. IIS is included with Microsoft's Windows NT Server 4.0 and above.
ITP	Interactive Transaction Processing. Extends the concept of OLTP to include Internet and intranet activities.
iTP	A Tandem trademark name for a product line that encompasses Internet security products, Internet server hardware products, and Internet application software.
ISV	Independent Software Vendor.
Java	A general purpose, object-oriented programming language, similar to C++ in some ways, developed by Sun Microsystems for the express purpose of developing lightweight and portable code.
JDBC	Java Database Connectivity.
MSCS	Microsoft Cluster Server. Formerly known by the code-name "Wolfpack."

MTS	Microsoft Transaction Server. Formerly known by the code-name “Viper,” MTS is Microsoft’s OLE/DCOM/ActiveX based transaction processing monitor and object request broker.
N+1	Describes Tandem’s approach to availability, which means providing one more of a given component than needed to keep the application running at the desired performance level. For example, if 14 processors are required for the application, a system consisting of 15 processors will allow for a processor failure.
node	A single Windows NT Server machine running NonStop Software. Formerly described as a “processor element” in the NonStop Himalaya architecture.
NetBIOS	Network Basic Input/Output System.
NonStop Cluster	A group of from 1 to 16 Windows NT Server nodes. Conceptually equivalent to a NonStop Kernel system consisting of from 1 to 16 CPUs.
NonStop tpX	Formerly known as OLE/TP. A client development tool and runtime environment that enables connectivity between Windows clients and TUXEDO and Pathway servers.
ODBC	Open Database Connectivity.
OLAP	Online Analytical Processing.
OLE	Originally known as “Object Linking and Embedding,” OLE is the foundation component technology that underlies Microsoft COM/DCOM architecture.
OLTP	Online Transaction Processing.
ORB	Object Request Broker.
OTM	Object Transaction Monitor.
OTS	Object Transaction Server.
RDO	Remote Data Objects.
RMI	Remote Method Invocation. A peer-to-peer API for distributed processing between Java servlets.
RPC	Remote Procedure Call.
RM	Resource Manager. An interface and associated software providing access to a collection of information and/or processes; for example, a DBMS. Resource managers provide transaction capabilities and permanence of actions, and are the entities accessed and controlled within a global transaction.

Resource manager instance	A particular instance or occurrence of a resource manager (for example, the EMPLOYEE database). There may be many occurrences or instances of the same or different resource managers within a global transaction, each managing different data. Each resource manager instance is considered to be autonomous, in full control of local access (for both local and global transactions), administration, and so forth.
RMI	Remote Message Invocation.
ROLAP	Relational Online Analytic Processing.
RPC	Remote Procedure Call.
SQL	Structured Query Language.
TDI	Transport Definition Interface.
TIP	Transaction Internet Protocol. An IETF standards track draft protocol specification being jointly developed by Tandem and Microsoft. TIP is a two-phase commit protocol designed for transaction processing between heterogeneous TP monitors.
/T	TUXEDO System /T. The main server-side run-time component of the TUXEDO distributed processing system.
X/Open	Founded in 1984, X/Open's brand mark is recognized worldwide as a guarantee of compliance to open systems specifications. X/Open is now part of The Open Group.
The Open Group	The Open Group was formed in February 1996 by the consolidation of the two leading open systems consortia, X/Open Company Ltd (X/Open) and the Open Software Foundation (OSF). Under the Open Group umbrella, OSF and X/Open work together to deliver technology innovations and wide-scale adoption of open systems specifications.
OSF	Open Software Foundation. As of February 1996, part of the The Open Group. Originally founded in 1988, OSF hosts industry-wide, collaborative, software research and development for the distributed computing environment (DCE).
Win32	Windows NT API.
Winsock	Windows Sockets API.

Figures

Tables