

Microsoft™ Enterprise Services

January 2005



Copyright

The information contained in this document represents the current view of Microsoft Corporation on the issues discussed as of the date of publication. Because Microsoft must respond to changing market conditions, it should not be interpreted to be a commitment on the part of Microsoft, and Microsoft cannot guarantee the accuracy of any information presented after the date of publication.

This White Paper is for informational purposes only. MICROSOFT MAKES NO WARRANTIES, EXPRESS, IMPLIED, OR STATUTORY, AS TO THE INFORMATION IN THIS DOCUMENT.

Complying with all applicable copyright laws is the responsibility of the user. Without limiting the rights under copyright, no part of this document may be reproduced, stored in or introduced into a retrieval system, or transmitted in any form or by any means (electronic, mechanical, photocopying, recording, or otherwise), or for any purpose, without the express written permission of Microsoft Corporation.

Microsoft may have patents, patent applications, trademarks, copyrights, or other intellectual property rights covering subject matter in this document. Except as expressly provided in any written license agreement from Microsoft, the furnishing of this document does not give you any license to these patents, trademarks, copyrights, or other intellectual property.

Unless otherwise noted, the example companies, organizations, products, domain names, e-mail addresses, logos, people, places, and events depicted herein are fictitious, and no association with any real company, organization, product, domain name, e-mail address, logo, person, place, or event is intended or should be inferred.

© 2005 Microsoft Corporation. All rights reserved.

Microsoft, Active Directory, BizTalk, InfoPath, JScript, Outlook, SharePoint, Visio, Visual Basic, Visual Studio, and Windows are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.

The names of actual companies and products mentioned herein may be the trademarks of their respective owners.

The Microsoft Enterprise Development Strategy Series

The Enterprise Development Strategy Series of briefing papers from Microsoft are designed to deliver overview and technical information of Microsoft application development technologies to developers, architects and IT management. The papers in this occasional series describe strategic goals Microsoft is addressing and summary technical information on these technologies and how they address the needs of the enterprise.

Feedback is important to us. If you have comments on the series, please email that information with a subject line of "Feedback" to entdevfb@microsoft.com.

Contents

Overview	1
Customer Problems.....	2
Data Consistency	2
Transacted Business Objects	2
Atomic Business Objects and Services.....	2
Transactions Across Multiple Atomic Business Objects	2
Transactions Across Multiple Databases	3
Leveraging Existing Technology Investments.....	3
Supporting Sometimes-Connected Services	3
Security.....	4
Design Goals for the Technology	5
Transaction Support Across Multiple Components.....	5
Automatic Transaction Processing.....	5
Bring Your Own Transaction (BYOT).....	5
Including Non-transactional Objects in a Transaction.....	5
Just-in-Time Activation	6
Loosely Coupled Events.....	6
Object Pooling	6
Role-Based Security.....	6
How Today's Technology Solves those Problems.....	8
Serviced Components	8
The Distributed Transaction Coordinator	8
XA Transactions	9
IBM CICS LU 6.2 Transactions	9
Scalability	9
Application Pooling.....	10
Availability.....	10
Manageability	11
Messaging	13

Overview

Today's IT world is heterogeneous. Even organizations that attempt to standardize on a single platform discover that they must interact with outside organizations that use disparate platforms. When building enterprise applications, organizations must take into account the various technologies with which they must interoperate. This may mean working with an entirely different hardware and operating system platform or working with multiple database engines. It may also mean the need to send messages from one platform to another.

Enterprise applications in such environments must support a variety of capabilities, one of which is the ability to ensure the consistency of data across different databases. It is not uncommon to need to update two different systems at once; for example, updating the order entry and inventory systems when a new order is placed. If the order entry system is a commercial package using Oracle and the inventory system is a custom application using SQL Server, organizations need a way to update both databases as a single transaction; failure to update one database should mean that changes to the other database are rolled back.

Not only must data remain consistent across different database platforms, but it must remain consistent across multiple, independently created business objects. Many large applications include a host of business objects that perform different functions. A single logical transaction may require the use of inventory, order, customer, and product objects, just to name a few. These objects are typically built to be as independent of each other as possible, which is almost always the right approach, but this greatly complicates data consistency over these objects during a single transaction.

Enterprise applications have another major interoperability concern: messaging. It is often necessary to pass messages between disparate systems. Thanks to Web services, applications can easily perform synchronous calls between different platforms, but asynchronous messaging with guaranteed delivery is a more challenging proposition. In situations in which companies want to provide messaging services across heterogeneous platforms or to systems that may only be occasionally available, there is a need for an infrastructure in place that is easily integrated with technologies from other companies and easily accessible to the developers of a company's internal applications.

It is entirely possible for an organization to create the underlying infrastructure to ensure data consistency across multiple databases, ensure data consistency across independently created business objects, and to allow for asynchronous messaging across disparate platforms. However, these problems are extraordinarily complex and the effort required to build them should not be underestimated. Fortunately, the solution is already provided by the Microsoft platform. The underlying architecture exists, and thanks to the .NET Framework, developers have easy access to this infrastructure. Building enterprise applications is made significantly easier by utilizing the existing technology.

Customer Problems

There are a number of technical requirements relating to the ability to manage transactions across databases and business objects, as well as messaging between applications and computers. These technical requirements represent challenging issues for IT departments to handle when designing and building enterprise applications. These challenges are detailed below and represent some of the greatest challenges to organizations building solutions that span applications, computers, platforms, and even extend to outside vendors.

Data Consistency

When completed, a transaction must leave all data in a consistent state. In a relational database, all rules must be applied to the transaction's modifications to maintain all data integrity. All internal data structures must be correct at the end of the transaction. This is simple enough in most single databases, as most vendors provide a transaction engine in the database. However, this does nothing to help applications span multiple databases, nor does it assist developers in maintaining state across separate business objects.

Transacted Business Objects

It is often necessary for a single business object to support transactions, as well as the need for a transaction to span multiple business objects. Integrating components with transactions can be a challenge as organizations attempt to tie into different transaction managers from different vendors. Clearly, a single transaction programming model can greatly simplify the work of making components work with transactions.

Atomic Business Objects and Services

The functionality provided by the components and services an organization creates should be atomic; in other words, they should do one thing and one thing only. Business objects should be as independent of each other as possible; this is known as loose coupling. Keeping an application's objects and services loosely coupled means that it is easier to reuse them in across applications, to modify them without affecting other parts of the application, and to ensure an easier understanding of their impact on data and other parts of the system. The challenge is that a single logical transaction, such as adding a new order, might touch a number of individual objects. Therefore, it becomes necessary to span that logical transaction across a number of objects that have no knowledge of each other.

Transactions Across Multiple Atomic Business Objects

Spanning transactions across multiple atomic business objects, which are created independently and loosely coupled from each other, requires more than simply tying into the transaction handling capabilities of a client database. Even coding the transactions into the separate business objects will typically not work, as database transactions are tied to

connections, and each object, in order to maintain atomicity, may make its own connection, and therefore manage its own transaction. This presents a great challenge because now a single logical transaction must become a physical transaction that encompasses the logic of these separate objects and allows a failure in any one object to roll back the changes in the other components, even though the failed object is completely unaware of the other objects participating in the transaction.

Transactions Across Multiple Databases

Whether a transaction is contained in a single business object or spans multiple business objects, data may need to be inserted, updated, or deleted in multiple databases. Coordinating these changes across heterogeneous databases presents a unique challenge, since each database vendor may provide their own unique transaction mechanism. Each database to be affected must allow for the data to be changed and have those changes roll back if any operation fails, and the data in each database must remain consistent at all times.

Leveraging Existing Technology Investments

Adding extensive support for distributed heterogeneous transactions and messaging across disparate platforms would be far less appealing if businesses had to throw out all existing platforms and software. Rather, they'd prefer to integrate with technologies and applications already in place. The ability to leverage an enterprise's existing infrastructure is critical when developing enterprise applications. Whether companies have existing Java transactions, CICS transactions, data in multiple databases, or messaging systems such as IBM's MQSeries, the ability to use these existing resources can greatly reduce the financial impact of implementing new applications.

Supporting Sometimes-Connected Services

Not all resources are constantly available, nor do they always need to be. When individuals buy an airline ticket online they can request a specific type of meal. However, the company providing the meals does not have to be notified immediately; the airline can issue the ticket while asynchronously notifying the food vendor about the special order. This notification is in the form of a message added to a queue. The food vendor can connect at will and retrieve queued messages.

Alternatively, a field engineer may travel to various locations diagnosing and fixing problems, and connectivity may not always be available. However, the engineer still needs to be able to use his or her laptop to fill out the details on the trouble ticket. The changes to the trouble ticket could be added to a queue to be automatically delivered once the engineer reconnects with the network.

Programming in the support for services or systems that are not always available is one of the great challenges in enterprise applications. It is not enough to just add messages to a queue; developers must ensure delivery and have a mechanism to handle any errors in delivery, or errors from the service that receives the message.

Security

As if handling a transaction across atomic components or separate databases were not difficult enough, the whole transaction must be secure, allowing the components to log into the database(s) with the proper credentials. This may require taking the credentials of the user and passing them along the entire chain and having the objects impersonate the user when they call the database(s). Communication from the client to the business objects, and from the business objects to the database, must also be secure.

Design Goals for the Technology

Transaction Support Across Multiple Components

If a company's business process will be invoking other business processes in the context of an atomic transaction, all the invoked business processes must ensure their operations participate in the existing transaction so that their operations will roll back if the calling business logic aborts. It should be safe to retry any atomic operation if it fails without fear of making data inconsistent. Developers can think of a transaction boundary as an area where, upon failure, all data is returned to the state it was in before the transaction started. Transactions across servers running Windows can be managed using Distributed Transaction Coordinator (DTC), which is used by .NET Enterprise Services (COM+). To manage distributed transactions in heterogeneous environments, Microsoft provides COM Transaction Integrator (COMTI) and Host Integration Server 2000.

Using the Enterprise Services classes, developers can create "serviced components" that take advantage of COM+ services. COM+ provides a services framework for component programming models deployed in an enterprise environment. Some of these services available include just-in-time (JIT) activation, synchronization, object pooling, transactions, and shared property management. In a disconnected environment typical of Web-based applications, additional COM+ services can use used such as loosely coupled events, queued components, and role-based security.

Automatic Transaction Processing

Automatic transaction processing is a service provided by COM+ that enables developers to configure a class at design time to participate in a transaction at run time. This makes it incredibly easy for developers to tap into the underlying infrastructure that allows for distributed transactions.

Bring Your Own Transaction (BYOT)

The COM+ BYOT feature allows COM+ components to set a preexisting Microsoft Distributed Transaction Coordinator (DTC) or Transaction Internet Protocol (TIP) transaction as the transaction property of a new component's context. This allows COM+ components to be associated with transactions whose lifetimes are controlled by a transaction processing monitor, object transaction service, or database management system. BYOT is also useful in integrating with transactions coordinated by TIP.

Including Non-transactional Objects in a Transaction

A compensating resource manager (CRM) is a service provided by COM+ that enables developers to include non-transactional objects in Microsoft Distributed Transaction Coordinator (DTC) transactions. Although CRMs do not provide the capabilities of a full resource manager, they do provide transactional atomicity (all-or-nothing behavior) and durability through the recovery log.

Just-in-Time Activation

The COM+ Just-in-Time (JIT) Activation service allows idle server resources to be used more productively. When a component is configured as JIT activated, COM+ can deactivate an instance of it while a client still holds an active reference to the object. The next time the client calls a method on the object, COM+ reactivates the object transparently to the client, just in time.

Loosely Coupled Events

The loosely coupled event model provided by COM+ supports late-bound events or method calls between the publisher and subscriber and the event system. Rather than repeatedly polling the server, the event system notifies interested parties as information becomes available.

Object Pooling

The COM+ Object Pooling service enables developers to reduce the overhead of creating each object from scratch. When an object is activated, it is pulled from the pool. When the object is deactivated, it is placed back into the pool to await the next request.

Role-Based Security

The .NET Framework provides mechanisms to integrate managed code with COM+ security services.

Traditional COM+ services such as distributed transactions, just-in-time activation, object pooling, and concurrency management are available to .NET components. With .NET, such services are referred to as Enterprise Services. They are essential for many middle-tier .NET components running within .NET Web applications. Enterprise Services applications use RPC authentication to authenticate callers. This means that unless developers have taken specific steps to disable authentication, the caller is authenticated using either Kerberos or NTLM authentication. Authorization is provided through Enterprise Services (COM+) roles, which can contain Microsoft® Windows® operating system group or user accounts.

A number applications use synchronous communication through remote procedure calls, while Web-based applications rely on HTTP. But many distributed applications need the non-blocking, asynchronous communication provided by message queuing. Windows Server 2003, using a component of COM+ Services called COM+ Queued Components (previously known as Microsoft Message Queue, or MSMQ), provides applications with exactly this kind of service in. With Queued Components, an application can send messages to another application without waiting for a response (in fact, the target application might not even be running). Those messages are sent into a queue, where they are stored until a receiving application removes them. If a response is expected, the sender can check a response queue at its leisure—there's no obligation to block waiting for a message. Message queuing is a flexible, reliable approach to communication, one that's appropriate for many kinds of applications.

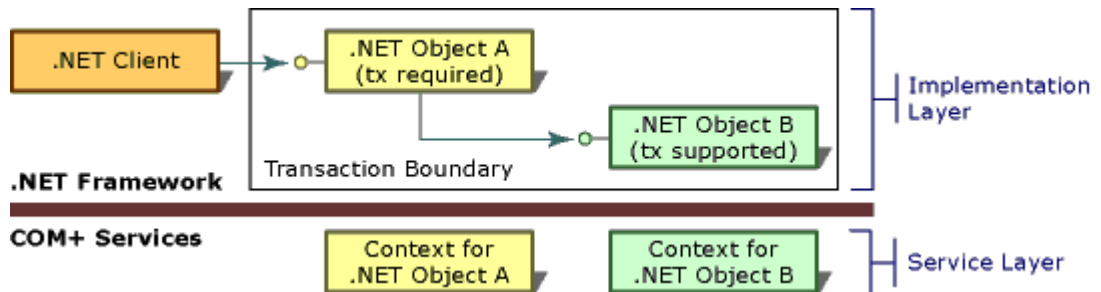
How Today's Technology Solves those Problems

Serviced Components

A serviced component is a class that is authored in a CLS-compliant language and that derives directly or indirectly from the System.EnterpriseServices.ServicedComponent class. Classes configured in this way can be hosted in a COM+ application and can use COM+ services by way of the EnterpriseServices namespace. For a list of supported services, see Summary of Available COM+ Services.

COM+ services, such as automatic transactions or Queued Components, can be configured declaratively. Developers apply service-related attributes at design time and create instances of classes that use those services. Services can then be configured by calling methods on service-related classes or interfaces. Some services can flow from one object to another. For example, an object configured to require a transaction can extend that transaction to a second object if the second object also supports or requires transactions.

The COM+ catalog holds the configuration information that is applied to a class implementation. At run time, based on the attributes applied to the code, COM+ creates a context service layer. The following illustration shows an automatic transaction that flows between two managed objects hosted by COM+.



The Distributed Transaction Coordinator

The Microsoft® Distributed Transaction Coordinator (the DTC) is a distributed transaction facility for Microsoft Windows® platforms which uses proven transaction processing technology. It is robust despite system failures, process failures, and communication failures; it exploits loosely coupled systems to provide scalable performance; and it is easy to install, configure, and manage. The DTC service provides the following benefits:

- Lowers the cost of enterprise computing—The DTC provides a sophisticated, low-cost distributed transaction facility for users of networked, commodity-priced PCs and servers.

- Simplifies application development—DTC transactions greatly simplify the application task of preserving consistency, despite failures that can occur when updating application data.
- Provides a consistent transaction model—The DTC supports a variety of resource managers, including relational databases, object-oriented databases, file systems, document storage systems, and message queues.
- Enables software development using distributed software components—The DTC provides a simple, object-oriented application programming interface for initiating and controlling transactions.

Using the DTC, applications gain the following performance advantages:

- Application programs can reliably update data residing in two or more resource managers, such as Microsoft SQL Server™ or Microsoft Message Queuing.
- Application programs can reliably update data residing in one or more XA-compliant resource managers, such as Oracle, IBM DB2, Informix, Sybase, or Ingres.
- Application programs can use an OLE Transactions-compliant resource manager, such as Microsoft SQL Server or Microsoft Message Queuing, with an X/Open-compliant transaction processing monitor such as Encina, TopEnd, or Tuxedo.
- Application programs can perform transactions spanning multiple transaction managers communicating via the Transaction Internet Protocol (TIP).
- Windows application programs can invoke IBM CICS transaction programs through the COM Transaction Integrator (COMTI).

XA Transactions

The DTC can act as either an XA-compliant resource manager or a transaction manager. When the DTC is acting as an XA-compliant resource manager, it allows Microsoft® SQL Server™, Microsoft Message Queuing, and other OLE Transactions-compliant resource managers to participate in transactions controlled by X/Open DTP XA-compliant transaction processing monitors such as Encina, TopEnd, and Tuxedo.

IBM CICS LU 6.2 Transactions

The DTC can participate in distributed transactions with IBM systems that support the IBM LU 6.2 protocol. If the connection between the Windows system and the IBM system fails and remains down, a transaction outcome can remain in doubt for an extended period of time. In this event, the IBM system may choose to heuristically commit or abort the transaction. Essentially, this means that the IBM system guesses the transaction's outcome rather than waiting to be told its outcome. Having heuristically committed or aborted the transaction, the IBM system can then release the locks held on behalf of the transaction. This allows other applications to access the records that were updated and locked by the transaction.

Scalability

Configurable Isolation Levels

One important Enterprise Services feature deals with transactions. Application developers can simply ask for a transaction without worrying about how many resource managers will be involved, or how the components involved in the transaction interact. Transactions ensure the integrity of the applications data by locking particular data records for a certain amount of time.

How much locking applications do, and for how long, is based on the isolation level. A higher isolation level means more locking, and consequently, a lower potential for incorrect data. Additionally, higher isolation levels also mean less concurrency and lower performance. A lower isolation level means less locking and more concurrency. Lower isolation levels also present a higher potential for incorrect data. Choosing a lower isolation level can result in improved performance but developers must be aware of the types of concurrency problems they can encounter based on the choice of isolation level.

Application Pooling

Enterprise Services server applications, running on Windows 2000, always run in a single process. There are certain types of legacy COM objects that are thread unaware and therefore must always run in the main thread of this process. The resulting effect is that the application does not scale very well because all requests must be queued to execute on the main thread. When running this type of application on Windows Server 2003, administrators can specify that a pool of processes will be used. This approach increases throughput in the same way that a thread pool can increase throughput in a single process. Additionally, having a pool of processes helps protect against process failures since each client request will be dispatched to the next process in the list.

Availability

Application Recycling

The performance of most applications degrades over time. This can be a result of memory leaks, reliance on third-party code, and non-scalable resource usage. Enterprise Services running on Windows Server 2003 provides process recycling as a simple solution to gracefully shut down a process and restart it. Process recycling significantly increases the overall stability of applications, offering a quick fix for known problems and a safeguard against unexpected ones.

Administrators can configure process recycling administratively through the component services UI, or programmatically through the COM+ administrative SDK. Administrators can shut down and recycle processes based on several criteria, including elapsed time, memory usage, the number of calls, and the number of activations.

Applications as NT Services

Developers using Enterprise Services and Windows Server 2003 now have the ability to configure an Enterprise Services server application as a Microsoft Windows NT service. Previously, only base COM applications could run as a Windows NT service.

This feature gives developers more control over when an Enterprise Services application starts. Marking it to run as a Windows NT service means that the application can be loaded into memory when the system boots. This is especially useful if organizations want to make their Enterprise Services application highly available by installing it on a clustered server. For improved security, administrators can also run a service under special built-in accounts designed for services such as NetworkService or LocalService. These accounts include the minimum privileges required to run as a service and do not require developers to change passwords when they expire.

Low-Memory Activation Gates

System administrators know it can be difficult to configure a server so that its resources are adequate for peak loads. When a server does not have enough physical resources to make peak demand, the server can exhaust virtual memory.

Exhausting virtual memory becomes a problem if the user code or system code does not properly handle memory allocation failures. The server begins to slow down, and as memory is exhausted, memory allocations fail. The server then executes error paths to handle the allocation failures. Finally, if an error path contains a bug in the system or user code running on the server, it is extremely difficult to trap and handle safely.

Enterprise Services, running on Windows Server 2003, prevents situations in which these error paths might be run on a server. Rather than wait for memory allocations to fail in a piece of code, Windows Server 2003 checks memory before it creates a new process or object. If the percentage of virtual memory available to the application falls below a fixed threshold, the activation fails before the object is created. By failing these activations that would normally run, the low-memory activation gates feature greatly enhances system reliability.

Web Services

Web services increases availability by making the service available to all kinds of clients. Enterprise Services uses .NET remoting to bring Web services to the world of COM by making it possible for any COM object to be accessed as a Web service, or to call a Web service as though it were any other COM object. Organizations can also use .NET remoting as a simple way to make a .NET class library accessible to Web-service clients by simply selecting a box, or adding an assembly attribute to the application. When registered, this approach will cause the Enterprise Services application to create a Web site on Microsoft Internet Information Server (IIS), and to connect the Web site to the components automatically.

Manageability

Pause or Disable Applications

When running on Windows 2000, it is not possible to prevent server applications from being activated. If the application is stopped and a request to activate a component from that application is received, the application will immediately start again. Administrators need to be able to pause or disable applications so they can make changes to the application

configuration, or assemblies and component DLLs, without removing the server from the network.

When running Enterprise Services on Windows Server 2003, administrators can pause any running process. While the process is paused, it will not accept requests for further activations. If requests are received, a new process will start. This enables administrators to troubleshoot applications by attaching a debugger or doing a process dump while the process is paused. Other clients will have their work routed to a new process so they can continue with their work.

Administrators can also disable an application or component so that activations will not be processed. If an application or component is disabled it will remain disabled until it is re-enabled, even if the server is rebooted. This allows administrators to disable an application in order to deploy a new version of the assemblies, or to make other changes to the application configuration.

Process Dump

It is not easy to troubleshoot applications in a production environment. How do developers gather information on a problem without disturbing the running processes? Enterprise Services running on Windows Server 2003 provides a solution through its new process dump feature. This feature allows the system administrator to dump the entire state of a process without terminating it.

For example, the application service provider (ASP) hosting an organization's application finds a problem, but cannot provide much more information other than the fact that something is not working. With the process dump feature, the ASP can take a non-invasive snapshot of the running processes. The details are saved in a log file, which the ASP can send to the organization or Microsoft Product Support Services, making it much easier to troubleshoot the Enterprise Services application. Using advanced debuggers like WinDBG, IT professionals can quickly identify which thread is causing the problem and then diagnose the problem.

Application Partitions

On Windows 2000, COM+ allows administrators to configure an implementation of a component only one time on a computer. When using Enterprise Services with Windows Server 2003, administrators can use a feature called application partitions that allows multiple versions of COM and .NET components to be installed and configured on the same machine. This feature can save organizations the cost and time-consuming effort of using multiple servers to manage different versions of an application.

To see the benefits, consider a hosted application. For example, if a provider has 1000 different customers accessing an application through an ASP. All 1000 customers require different back-end database and different security settings. To do this with Windows 2000, providers would have to install 1000 versions of the application on 1000 physically separate servers.

With the application partitions feature in Enterprise Services, providers or organizations can create 1000 partitions, one for each version of the application, on a single computer. Each partition acts, in effect, as a virtual server. After installing the application into each partition, organizations can create partition sets that map users to the logical servers. This mapping determines the application version that customers will access. Users can only access components from partitions in their partition sets. The benefit of partitions is clear. It is much easier and more cost-effective to manage one large server, or even a few larger servers, rather than many small servers.

HIS

An estimated 70 percent of all corporate data is stored on host systems, such as IBM mainframe and AS/400 computers. Yet, increasingly, organizations rely on personal computers together with Web-based and Windows®-based applications for everyday productivity and line-of-business solutions. Companies have discovered that Web and Windows solutions often are easier to learn and quicker to implement than comparable host-based applications. To preserve their time and capital investments in host technology, organizations must either migrate all of their host-based resources to the Windows platforms, which can be expensive and time-consuming, or integrate their host-based resources with more efficient Windows-based and Web-based solutions.

When developing transaction-aware applications for the Windows and Web platforms, developers often write COM components that run under the Microsoft Transaction Server (MTS) in Windows NT or the equivalent COM+ feature in Windows 2000. MTS combines the features of a COM object broker and a transaction manager. An application can be written with COM components that run on different computers. The programmer can define how each component participates in transactions across these distributed components. Each COM component can specify whether it can be part of a new transaction or an existing transaction. The transaction management part of MTS is based on a component called the Distributed Transaction Coordinator (DTC). Once the transaction state of a set of components is defined, then MTS can use the DTC to enforce the same transaction and database update integrity over them that a CICS or IMS program can enforce over mainframe transactions. COM programs can indicate the success or failure of a complete transaction just as a CICS or IMS program can do.

In order to allow applications for Windows NT and Windows 2000 to make use of these pre-existing mainframe CICS and IMS programs, Host Integration Server 2000 offers COM Transaction Integrator (COMTI). COMTI enables mainframe CICS and IMS programs to participate in COM transactions.

Messaging

Message Queuing (MSMQ) technology enables applications running at different times to communicate across heterogeneous networks and systems that may be temporarily offline. Applications send messages to queues and read messages from queues. Message Queuing provides guaranteed message delivery, efficient routing, security, and priority-based messaging.

It can be used to implement solutions to both asynchronous and synchronous scenarios requiring high performance. The following list shows several places where Message Queuing can be used.

- Mission-critical financial services: for example, electronic commerce.
- Embedded and hand-held applications: for example, underlying communications to and from embedded devices that route baggage through airports by means of an automatic baggage system.
- Outside sales: for example, sales automation applications for traveling sales representatives.
- Workflow: Message Queuing makes it easy to create a workflow that updates each system. A typical design pattern is to implement an agent to interact with each system. Using a workflow-agent architecture also minimizes the impact of changes in one system on the other systems. With Message Queuing, the loose coupling between systems makes upgrading individual systems simpler.

Interoperability

Message Queuing provides support for interoperability with other messaging systems. With the MSMQ-MQSeries Bridge, a connector application available in Microsoft Host Integration Server 2000, Message Queuing can be used to communicate with IBM MQSeries systems. Sending applications can send messages to destination queues, referred to as foreign queues, in the foreign messaging system as if they were destination queues within a company's own Message Queuing enterprise.

References

Transactions (MSDN Library)

http://msdn.microsoft.com/library/default.asp?url=/library/en-us/acdata/ac_8_md_06_2it2.asp

Message Queuing (MSMQ)

http://msdn.microsoft.com/library/default.asp?url=/library/en-us/msmq/msmq_overview_4ilh.asp

Enterprise Services Security

<http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnnetsec/html/SecNetch09.asp>

O COM+, Where Art Thou?

<http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnadvnet/html/vbnet04232002.asp>

Application Architecture for .NET: Designing Applications and Services

<http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnbda/html/AppArchCh2.asp>

A Guide to Building Enterprise Applications on the .NET Framework

<http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnbda/html/guidenetapp.asp>

Summary of Available COM+ Services

<http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpguide/html/cpconsummaryofservices.asp>

Building Secure ASP.NET Applications: Authentication, Authorization, and Secure Communication by J.D. Meier, Alex Mackman, Michael Dunner, and Srinath Vasireddy

<http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnnetsec/html/SecNetch09.asp>

Serviced Component Overview

<http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpguide/html/cpconservicedcomponentoverview.asp>

Windows Server 2003 and Enterprise Services by Ron Jacobs

http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnnetsec/html/WINDotNETSvr_dotNETEntpSvs.asp

Microsoft Host Integration Server 2000 Product Overview

http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnhis/html/his_hiserver2000.asp

.NET Enterprise Services and COM+ 1.5 Architecture

<http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnentsrv/html/netenterpriseandcomplus.asp>

Using Microsoft Transaction Server 2.0 with COM Transaction Integrator 1.0

http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnmts/html/msdn_comti.asp

Summary of Available COM+ Services

<http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpguide/html/cpconsummaryofservices.asp>

Designing the Components of an Application or Service

<http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnbda/html/AppArchCh2.asp>

UNIX and Windows Compared

<http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnucmg/html/UCMGch02.asp>

A Guide to Building Enterprise Applications on the .NET Framework

<http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnbda/html/guidenetapp.asp>

An Overview of Transaction Processing Concepts and the MS DTC

http://msdn.microsoft.com/archive/default.asp?url=/archive/en-us/dnarsqlsg/html/msdn_dtcwp.asp

MSMQ Info

http://msdn.microsoft.com/library/default.asp?url=/library/en-us/msmq/msmq_overview_4ilh.asp

Serviced Component Overview

<http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpguide/html/cpconservicedcomponentoverview.asp>

Windows Server 2003 and Enterprise Services

http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnnetsev/html/WINdotNETSvr_dotNETEntpSvs.asp

Using COM+ Services in .NET

<http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dndotnet/html/comservnet.asp>

Chapter 17 – Securing your Application Server

<http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnnetsec/html/THCMCh17.asp>

Building Secure ASP.NET Applications: Authentication, Authorization, and Secure Communication

<http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnnetsec/html/SecNetch05.asp>

<http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnnetsec/html/SecNetch07.asp>

Implementing Database Transactions with Microsoft .NET

<http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnbda/html/psent.asp>

.NET Data Access Architecture Guide

<http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnbda/html/daag.asp>

COM+ Integration: How .NET Enterprise Services Can Help You Build Distributed Applications

<http://msdn.microsoft.com/msdnmag/issues/01/10/complus/default.aspx>

Microsoft Host Integration Server 2000 Product Overview

http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnhis/html/his_hiserver2000.asp

<http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnaspp/html/aspnetscal.asp>